НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.4'4

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Системне програмування

на тему: Організація розподілених обчислень при відстеженні траєкторії об'єктів у мережі сенсорів

Виконав: студент II курсу, групи КВ-63м

Перевертайло Андрій Ігорович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник доцент, к.т.н, Замятін Д.С._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)
«___»_____2018р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Перевертайло Андрій Ігорович
(прізвище, ім'я, по батькові)

1. Тема дисертації: ОРГАНІЗАЦІЯ РОЗПОДІЛЕНИХ ОБЧИССЛЕНЬ ПРИ ВІДСТЕЖЕННІ ТРАЕКТОРІЇ ОБ'ЄКТІВ У МЕРЕЖІ СЕНСОРІВ,
науковий керівник дисертації Замятін Денис Станіславович, к.т.н., доцент____ ,
затверджені наказом по університету від «22» березня 2018 р. №986-с

2. Термін подання студентом дисертації 11 травня 2018 р.

3. Об'єкт дослідження: методи проектування системи розподілених обчислень у сенсорних мережах при локалізації джерел акустичного сигналу.

4. Предмет дослідження: реалізація системи розподілених обчислень у сенсорних мережах при локалізації джерел акустичного сигналу.

5. Перелік завдань, які потрібно розробити:
- розробка моделі, що імітує локалізацію джерел акустичних сигналів в сенсорній мережі та складається з мережі сенсорів та серверів, що утворюють кластер, а також розробка алгоритму, що дозволить реалізувати метод TDOA (Time Difference of Arrival) для локалізації сигналів та розподілити обчислення між серверами кластера використовуючи платформу Apache Spark.
- аналіз отриманих результатів

6. Перелік ілюстративного матеріалу: схема роботи розробленої моделі, загальний алгоритм роботи моделі, алгоритм реалізації методу TDOA, алгоритм обчислень на вузлах кластеру, UML діаграма класів, схема трансформації еластичних наборів даних.

7. Перелік публікацій:
- ПМК 2018 «Організація розподілених обчислень при локалізації об’єктів за акустичним сигналом у системі сенсорів»,
- САІТ 2018 «Розподілені обчислення координат місцезнаходження об’єкта у системі сенсорів»

8. Дата видачі завдання 5 вересня 2016 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	21.11.2016	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	11.04.2017	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	27.05.2017	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації	04.10.2017	
5.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	11.12.2017	
6.	Проведення наукового дослідження; підготовка матеріалів доповіді на конференції ПМК-2018	10.02.2018	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	05.04.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	20.04.2018	
9	Попередній розгляд магістерської дисертації на кафедрі	26.04.2018	

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. Акустичні сенсорні мережі все частіше використовуються в багатьох прикладних областях людської діяльності. Також продуктивнішою стає апаратна складова обчислювального ресурсу таких мереж. Разом з цим збільшуються вимоги щодо швидкодії та граничної кількості об’єктів, які мають бути одночасно локалізовані у таких мережах, тому питання застосування нових алгоритмів для забезпечення розподілених обчислень на вузлах мережі є актуальним.

Об’єктом дослідження є методи проектування системи розподілених обчислень у сенсорних мережах при локалізації джерел акустичного сигналу.

Предметом дослідження є реалізація системи розподілених обчислень у сенсорних мережах при локалізації джерел акустичного сигналу.

Мета роботи: розробка моделі, що імітує локалізацію джерел акустичних сигналів в сенсорній мережі та складається з мережі сенсорів та серверів, що утворюють кластер, а також розробка алгоритму, що дозволить реалізувати метод TDOA (Time Difference of Arrival) для локалізації сигналів та розподілити обчислення між серверами кластера використовуючи платформу Apache Spark.

Наукова новизна:

1. Розроблено метод для локалізації джерел акустичних сигналів в сенсорній мережі що реалізує метод TDOA (Time Difference of Arrival) для локалізації сигналів та забезпечує розподілення обчислень між серверами кластера використовуючи платформу Apache Spark.
2. Розроблено програмну реалізацію методу TDOA, що дозволяє ефективно обробляти дані сенсорів паралельно на визначеній кількості серверів на платформі Apache Spark.

Практична цінність отриманих в роботі результатів полягає в тому, що розроблений спосіб може бути застосований у нових або існуючих

системах локалізації джерела акустичного сигналу для збільшення швидкодії та кількості джерел акустичного сигналу. У свою чергу, розроблена програмна реалізація запропонованих способів може бути використана для вирішення реальних задач локалізації джерел акустичного сигналу.

Апробація роботи. Основні положення і результати роботи будуть представлені та обговорюватимуться на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 21-23 березня 2018 р.).

Структура та обсяг роботи. Магістерська робота складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуто загальні положення методу TDOA локалізації акустичного сигналу, класифікацію розподілених обчислювальних систем, проаналізовані платформи для організації розподілених систем.

У другому розділі розглядається реалізація методу TDOA на платформі Apache Spark.

У третьому розділі описане тестування створеної системи.

У четвертому розділі описаний аналіз роботи системи.

У висновках представлені результати проведеної роботи.

Робота представлена на 80 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: розподілені сенсорні мережі, локалізація, Apache Spark, Apache Hadoop, TDOA, Apache Hive, Apache Pig, Apache HBase, HDFS.

ABSTRACT

Topicality. Acoustic sensory networks are increasingly used in many applications of human activity. Also, the hardware component of the computing resource of such networks becomes more productive. At the same time, requirements for the speed and the maximum number of objects that must be simultaneously localized in such networks increase, therefore the question of applying new algorithms for providing distributed computing on the network nodes is relevant.

The object of study are methods of designing distributed computing systems in sensory networks when locating sources of acoustic signal.

The study examines the implementation of the distributed computing system in sensory networks in the localization of sources of acoustic signal.

Objective: to develop a model simulating the localization of acoustic signal sources in the sensor network and consists of a network of sensors and servers forming a cluster, as well as the development of an algorithm that will allow the time difference of the Arrival (TDOA) to implement the signal localization and distribute computations between cluster servers using Apache Spark platform.

Scientific innovation is as follows:

1. The method for localization of acoustic signal sources in the sensor network that implements the Time Difference of Arrival (TDOA) for signal localization is developed and provides distribution of computing between cluster servers using the Apache Spark platform.
2. Software implementation of the TDOA method has been developed, which allows efficient processing of sensor data in parallel on a certain number of servers on the Apache Spark platform.

Practical value obtained in the work of the results is that the developed method can be applied in new or existing systems of localization of the source of the acoustic signal to increase the speed and number of sources of acoustic signal. In turn, the program

implementation of the proposed methods can be used to solve real problems of localization of sources of acoustic signal.

Testing of work. The main provisions and results will be presented and discussed at a scientific conference undergraduates and graduate students “Applied mathematics and computing”, AMC-2017 (Kyiv, 21-23 March 2018).

The structure and scope of work.

The master's thesis consists of an introduction, four chapters and conclusions.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work, provides information on the approbation of the results and their implementation. *The first section* general provisions of the method of TDOA localization of the acoustic signal, the classification of distributed computing systems, analyzed platforms for the organization of distributed systems.

The second section discusses the implementation *of the TDOA method on the Apache Spark platform.*

The third section describes the testing of the system.

The fourth section describes the analysis of the system.

The conclusions are the results of the work.

Work submitted 80 pages, containing a link to a list of used literature.

Keywords: Distributed Sensor Networks, Localization, Apache Spark, Apache Hadoop, TDOA, Apache Hive, Apache Pig, Apache HBase, HDFS.

.

РЕФЕРАТ

Актуальность темы. Акустические сенсорные сети все чаще используются во многих прикладных областях человеческой деятельности. Также более продуктивной становится аппаратная составляющая вычислительного ресурса таких сетей. Вместе с этим увеличиваются требования по быстродействию и предельному количеству объектов, которые должны быть одновременно локализованы в таких сетях, поэтому вопрос применения новых алгоритмов для обеспечения распределенных вычислений на узлах сети является актуальным.

Объектом работы являются методы проектирования системы распределенных вычислений в сенсорных сетях при локализации источников акустического сигнала.

Предметом работы является реализация системы распределенных вычислений в сенсорных сетях при локализации источников акустического сигнала.

Цель работы: разработка модели, имитирующей локализацию источников акустических сигналов в сенсорной сети и состоит из сети сенсоров и серверов, образующих кластер, а также разработка алгоритма, который позволит реализовать метод TDOA (Time Difference of Arrival) для локализации сигналов и распределить вычисления между серверами кластера используя платформу Apache Spark.

Научная новизна заключается в следующем:

1. Разработан метод для локализации источников акустических сигналов в сенсорной сети реализующей метод TDOA (Time Difference of Arrival) для локализации сигналов и обеспечивает распределения вычислений между серверами кластера используя платформу Apache Spark.

2. Разработана программная реализация метода TDOA, что позволяет эффективно обрабатывать данные сенсоров параллельно на определенном количестве серверов на платформе Apache Spark.

Практическая ценность полученных в работе результатов заключается в том, что разработан способ может быть применен в новых или существующих системах локализации источника акустического сигнала для увеличения быстродействия и количества источников акустического сигнала. В свою очередь, разработана программная реализация предложенных способов может быть использована для решения реальных задач локализации источников акустического сигнала.

Апробация работы. Основные положения и результаты работы будут представлены и будут обсуждаться на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2017 (Киев, 21-23 марта 2018).

Структура и объем работы. Магистерская работа состоит из введения, четырех глав и выводов.

Во *введении* представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы, приведены сведения об апробации результатов и их внедрение.

В первом разделе рассмотрены общие положения метода TDOA локализации акустического сигнала, классификацию распределенных вычислительных систем, проанализированы платформы для организации распределенных систем.

В *первом разделе* рассмотрены общие положения метода TDOA локализации акустического сигнала, классификацию распределенных

вычислительных систем, проанализированы платформы для организации распределенных систем.

Во втором разделе рассматривается реализация метода TDOA на платформе Apache Spark.

В третьем разделе описано тестирования созданной системы.

В четвертом разделе описан анализ работы системы.

В выводах представлены результаты проведенной работы.

Работа представлена на 80 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: распределены сенсорные сети, локализация, Apache Spark, Apache Hadoop, TDOA, Apache Hive, Apache Pig, Apache HBase, HDFS.

Зміст

ПЕРЕЛІК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

SOSUS - американська мережа підводних акустичних детекторів (гідрофонів чи пасивних сонарів) для реєстрації переміщень підводних човнів.

SISVIA - проект з охорони навколишнього середовища.

TDOA - це метод, що дозволяє знайти місцеположення джерела сигналів за рахунок використання різниці в часі надходження звуку з джерела до сенсорів мережі.

APACHE SPARK - це кластерна обчислювальна система з відкритим кодом.

ZigBee - бездротовий стандарт передачі даних.

WiFi - це технологія бездротової локальної мережі з пристроями на основі стандартів IEEE 802.11.

IEEE - міжнародна організація інженерів у галузі електротехніки, радіоелектроніки та радіоелектронної промисловості.

APACHE Hadoop - програмна платформа з відкритим кодом і каркас для організації розподіленого зберігання і обробки наборів великих даних.

MapReduce - це програмна модель і програмний каркас, що її реалізує, впроваджена компанією Google для проведення розподіленої паралельної обробки великих масивів даних з використанням кластерів звичайних недорогих комп'ютерів.

Map - функція, яка виконує обробку пари ключ/значення і генерує набір проміжних пар ключ/значення.

Reduce- функція, яка з’єднує разом всі проміжні значення пов'язані однаковим проміжним ключем.

Yarn – платформа, яка відповідає за керування обчислювальними ресурсами в кластерах і їх використання для користувацьких задач.

HDFS - це розподілена файлова система, яка забезпечує високошвидкісний доступ до даних.

Java - об'єктно-орієнтована мова програмування.

Scala - мультипарадигмова мова програмування, що поєднує в собі як властивості об'єктно-орієнтованого так і функціанального програмування.

GFS - розподілена файлова система, яка впроваджена компанією Google в 2000 році.

Mesos - це централізована відмовостійка система управління кластером.

JDBC - прикладний програмний інтерфейс Java, який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до бази даних.

RDD - структура даних, яка являє собою пружні розподілені набори даних (англ. resilient distributed dataset, RDD), відмовостійка розподілена між кластером серверів множина елементів даних, яку можна лише читат

ВСТУП

Інтерес до вивчення розподілених сенсорних мереж обумовлений широкими можливостями їх застосування. Сенсорні мережі можуть бути використані в багатьох прикладних областях, таких як: системи захисту та забезпечення безпеки, спостереження навколишнього середовища, моніторинг промислового устаткування, охоронні системи, моніторинг стану сільськогосподарських угідь, моніторинг фізіологічного стану людини, контроль персоналу.

Важливою задачею у сенсорних мережах є організація розподілених обчислень даних сенсорів між обчислювальними вузлами, що утворюють кластер. Така організація має ряд переваг порівняно з мережами, що використовують один сервер для обчислень, таких як: масштабованість – динамічне додавання або видалення обчислювальних вузлів, паралельність – декілька процесів можуть одночасно виконуватися на різних вузлах, а також забезпечення відмовостійкості – резервування.

TDOA (Time Difference of Arrival) є одним з широко використовуваних методів локалізації, в якій ціль (джерело) випускає сигнал, а ряд анкерів (сенсорів) записує час виходу вихідного сигналу. Розраховуючи різницю часу різних приймачів, оцінюється розташування цілі. У такій схемі приймачі повинні бути точно синхронізовані.

APACHE SPARK - це кластерна обчислювальна система з відкритим кодом розроблена в Каліфорнійському університеті, AMPLab у Берклі. APACHE SPARK забезпечує інтерфейс для програмування цілих кластерів з неявним паралелізмом даних та відмовостійкістю.

В даній роботі реалізований алгоритм для локалізації об’єктів у розподілених сенсорних мережах, який базується на методі TDOA (Time Difference of Arrival) та

забезпечує розподіл обчислень на вузлах кластера використовуючи платформу APACHE SPARK.

1. ОПИС АРХІТЕКТУРИ АКУСТИЧНИХ СЕНСОРНИХ МЕРЕЖ ТА ОБГРУНТУВАННЯ МАГІСТЕРСЬКОЇ РОБОТИ

1.1 Огляд відомих проектів з використанням сенсорних мереж

Серед відомих проектів з використанням акустичних сенсорних мереж можна навести проект SOSUS (Sound Surveillance System) (рисунок 1). Проект являє собою підводну мережу моніторингу, яка складається із набору акустичних сенсорів, встановлених на дні океану, яку вважають однією із перших прототипів сенсорної мережі [2].

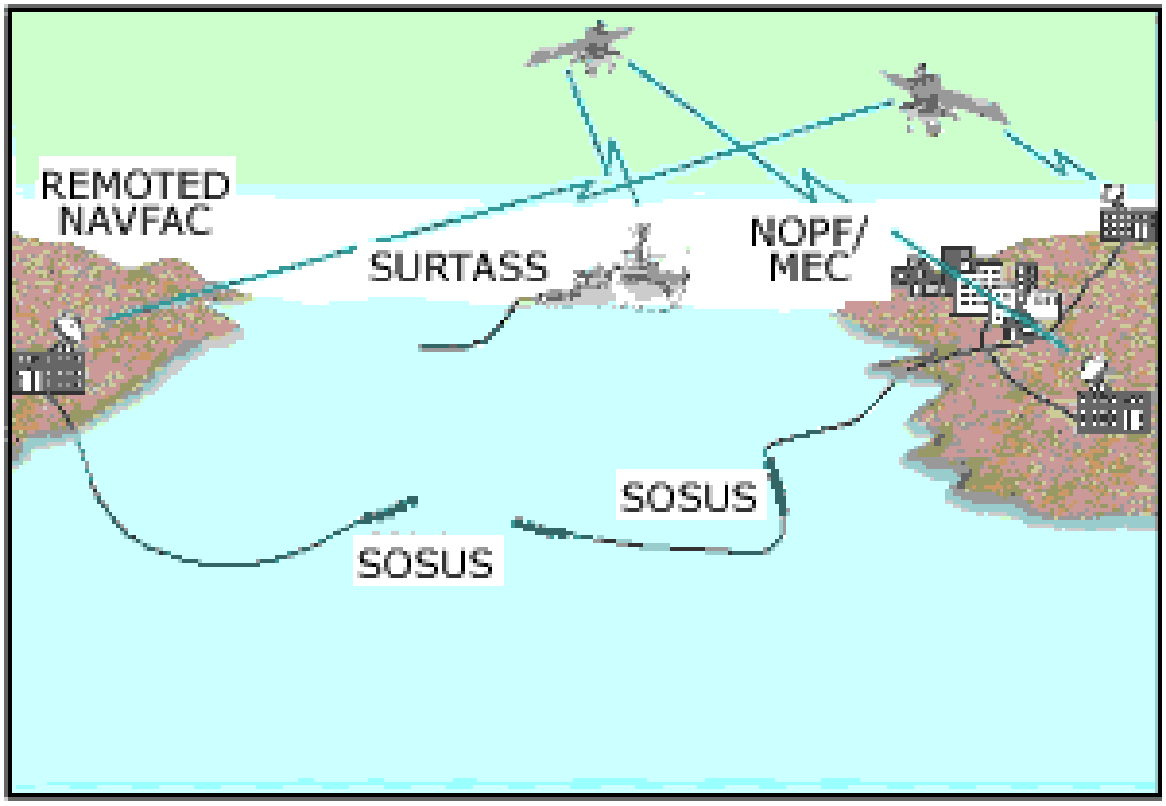

Рисунок 1 – Мережа моніторингу SOSUS

Ще один відомий проект - SISVIA (Sistema de Seguimiento y VIgilancia Ambiental) проект з охорони навколишнього середовища, був розроблений та впрваджений для моніторингу лісових пожеж за допомогою сенсорної мережі (рисунок 2) [3]. Площею близько 210 га в Іспанії.

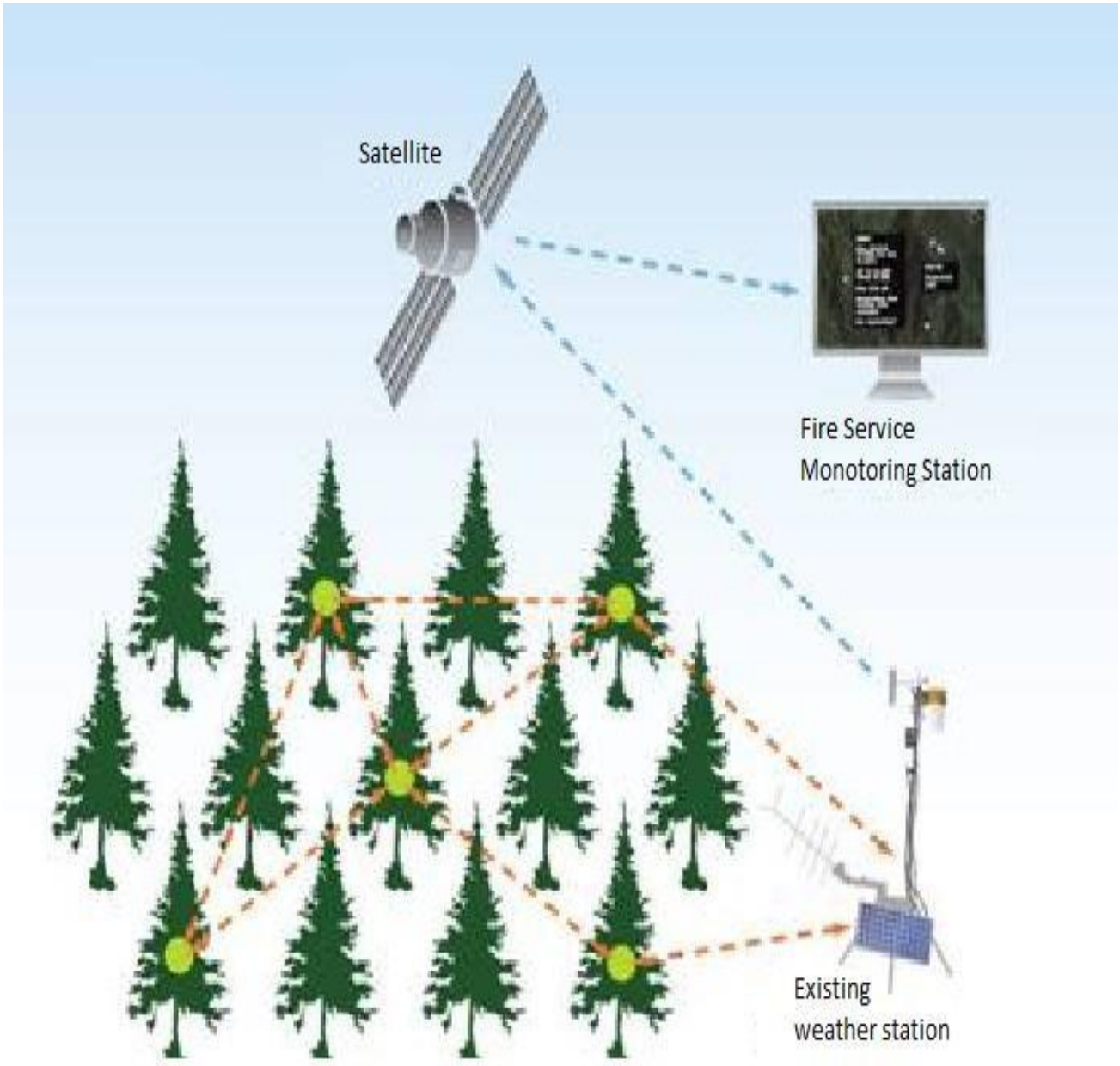

Рисунок 2 – Сенсорна мережа моніторингу лісових пожеж

Отже акустичні сенсорні мережі все частіше використовуються в багатьох прикладних областях людської діяльності. Також продуктивнішою стає апаратна складова обчислювального ресурсу таких мереж. Разом з цим збільшуються вимоги щодо швидкодії та граничної кількості об’єктів, які мають бути одночасно локалізовані у таких мережах, тому питання застосування нових алгоритмів для забезпечення розподілених обчислень на вузлах мережі є актуальним.

1.2 Загальна архітектура сенсорних мереж

Акустична сенсорна мережа (рисунок 3) - це самоорганізована розподілена мережа певної кількості сенсорів (мотів - від англ. "mote" - порошинка, названих так через невеликий розмір сенсора) (рисунок 4) і пристроїв обробки даних, з’єднаних між собою за допомогою мережі Інтернет. Максимальна поверхня покриття такої мережі до декількох кілометрів за рахунок здатності передачі даних від одного вузла до іншому.

В склад вузлів сенсорної мережі зазвичай входять автономні мікрокомп'ютери (контролери) з живленням від батарей і приймачі, що дозволяє мотам самоорганізовуватися в спеціалізовані мережі, зв'язуючись один з одним і обмінюючись даними (Рисунок 5).

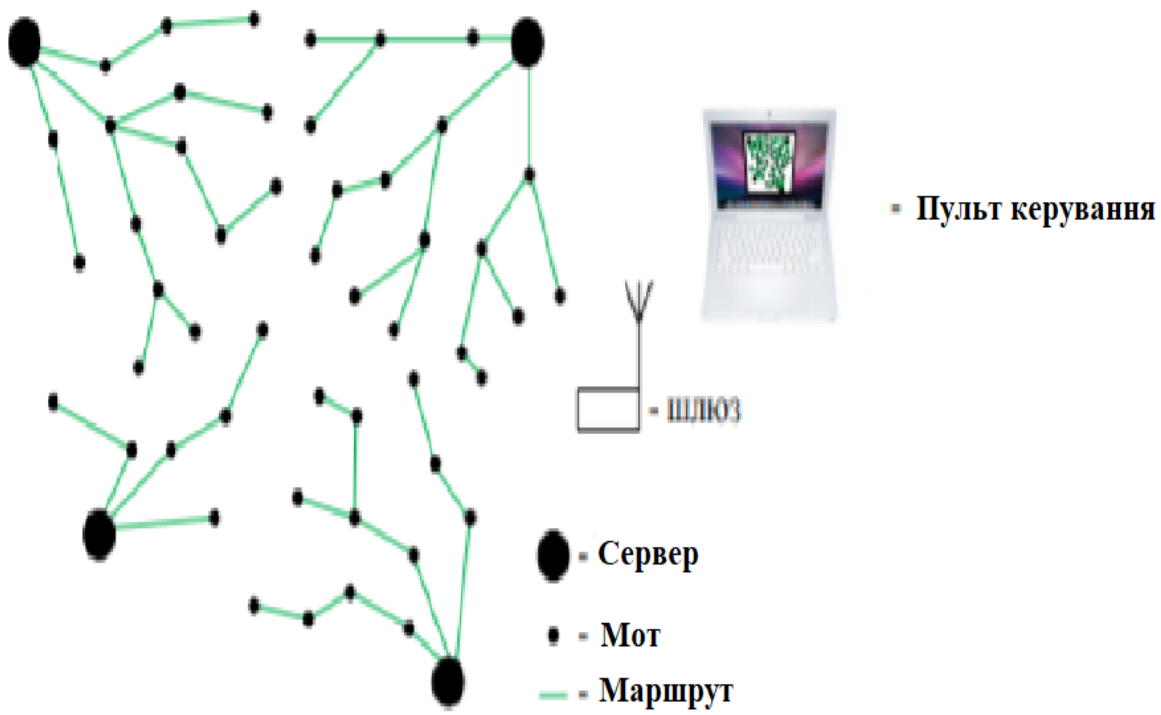

Рисунок 3 – Акустична сенсорна мережа

Рисунок 4 – Зовнішній вигляд акустичного сенсора

В цьому випадку, вузли сенсорної мережі виступають як компоненти бездротових сенсорних мереж. Апаратно вузол сенсорної мережі являє собою плату розміром, який не більше десяти кубічних сантимерів. На платі розташовані процесор, пам'ять - дискова і оперативна, цифроаналогові та аналого-цифрові перетворювачі, акустичний пристрій, джерело живлення та датчики.

Сенсори можуть мати різну структуру; вони утвоюють звязок через цифрові й аналогові конектори. Часто використовуються сенсори температури, тиску, вологості, освітленості, вібрації, рідше - магнітоелектричні, хімічні (наприклад, що вимірюють коцентрацію певних речовин), звукові та інші. Набір застосовуваних датчиків залежить від функцій, що виконуються безпроводними сенсорними мережами. Живлення мота здійснюэться от невеликої батареї.

Сенсори використовуються тільки для збору, первичної обробки та передачі сенсорних даних.

Дані від окремих вузлів передаються по мережі від одного вузла до іншого на шлюз, і зазвичай надходять на «супер-вузол», або сервер, що має більшу обчислювальну здатність.

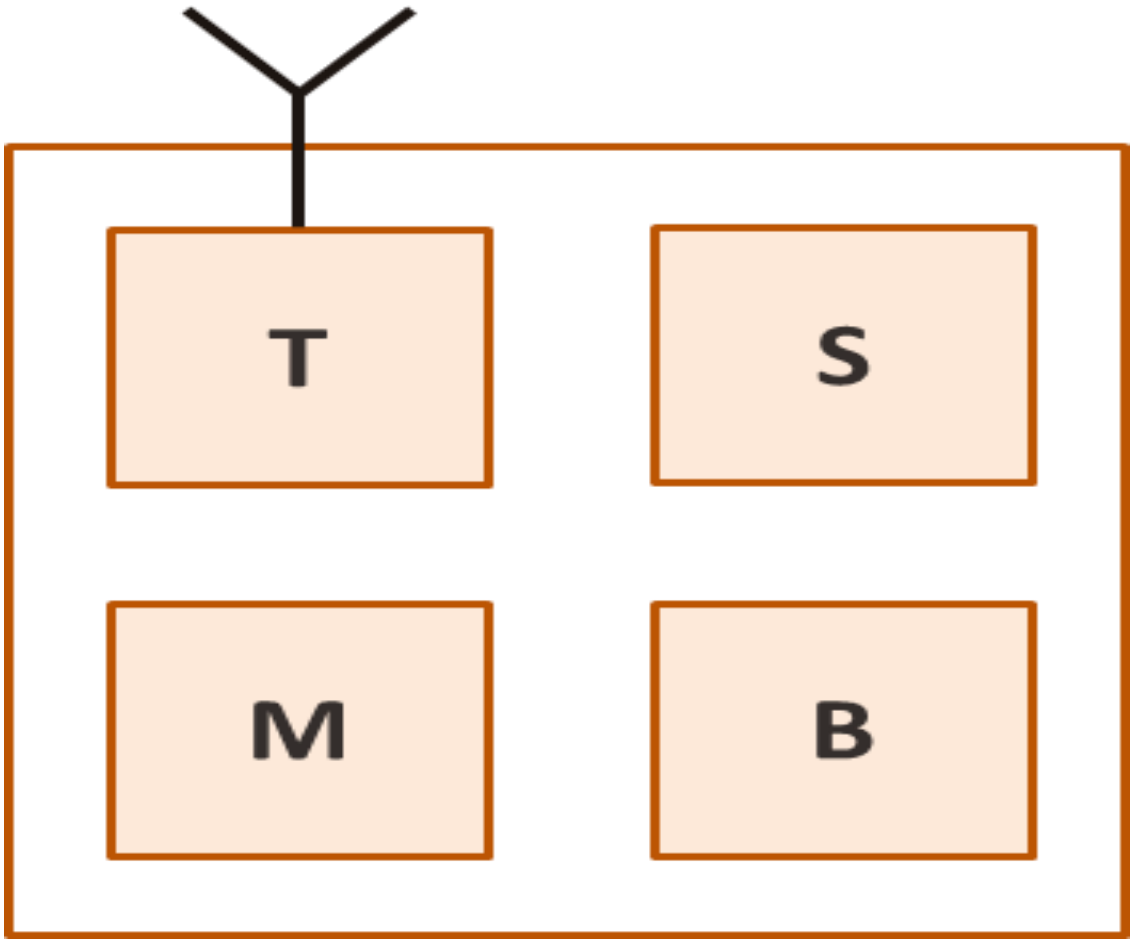

Рисунок 5 – Структура мота: T - Радіочастотний трансівер;

M – мікроконтролер; B – джерело живлення; S – сенсор.

Суттєвим обмеженням при роботі бездротових сенсорних мереж є обмежений заряд батарей, що встановлюються на сенсори. Варто враховувати, що заміна батареї як правило неможлива. У зв'язку з цим важливо виконувати на сенсорах тільки обмежену кількість дій, орієнтовану на зменшення кількості переданих

даних, а також мінімізувати кількість прийомів та передачі даних. З метою вирішення такого завдання було розроблено спеціальні комунікаційні протоколи, найбільш відомими з яких є протоколи альянсу ZigBee, які використовують раніше розроблений стандарт IEEE 802.15.4.

Стандарт IEEE 802.15.4 описує структуру лише двох нижчих рівнів моделі взаємодії (рисунок 6) : фізичний рівень (PHY) рівень управління доступом до каналу для трьох неліцензійних діапазонів частот: 2,4 Ггц, 868 Мгц і 915 Мгц . У таблиці 1 наведені основні властивості обладнання, що функціонує в цих діапазонах частот і канальний рівень (MAC), що відповідає за керування доступом до каналу з використанням методу множинного доступу з розпізнаванням несучої частоти й видаленням колізій (Carrier Sense Multiple Access with Collision Avoidance, CSMACA), а також за управлінням підключення і відключення від мережі передачі даних і забезпечення захисту переданих даних симетричним ключем шифрування AES-128.

Таблиця 1 – Властивості передачі даних для IEEE 802.15.4

Діапазон частот, Мгц	Швидкість передачі даних, Кбіт/с	Число каналів	Регіон
868,3	20	1	Європа
902-928	40	1-10	Америка
2405-2480	250	11-26	Весь світ

Бездротовий стандарт IEEE 802.15.4 може здійснювати обмін даними у 27 каналах в 3 частотних діапазонах (868 і 915 Мгц, а також 2,4 ГГц). Це дає можливість забезпечити ліцензійне застосування стандарту на території більшості країн (як відомо, діапазони, доступні для цивільного використання, визначають урядом залежно від поглядів на це питання), а також забезпечити якісну передачу сигналу в різних умовах. В Україні швидкість такої передачі в дозволеному частотному діапазоні 2,4 ГГц досягатиме 250 кбіт в секунду (у інших діапазонах вона істотно нижча).

Співвідношення "сигнал/завади" дозволяють сигналам стандарту IEEE 802.15.4 успішно працювати в середовищі з іншими джерелами випромінювання на тій же частоті, наприклад, вузлами, сполученими за допомогою WiFi.

Компанії, що виготовляют технічні засоби у широкому спектрі галузей промисловості, включаючи промислову автоматику і проектування, комунальні служби електропостачання, технології, виробництво об'єдналися в Альянс ZigBee для того, щоб спільно сформувати уніфікованиий протокол для бездротової передачі даних на основі специфікації ZigBee і сприяти її використанню. Зараз до складу Альянсу ZigBee входять більше 225 компаній. У їх числі:

Ради Директорів: Cellnet, Eaton Corporation, Ember Corporation, Freescale Semiconductor, Honeywell, Huawei Technologies, Itron, Mitsubishi Electric, Motorola, Philips, Samsung, Schneider Electric, Siemens, STMicroelectronics, Tendril Networks і Texas Instruments.

ZigBee активно впрваджуються сімома провідними постачальниками напівпровідникового устаткування: членами Альянсу є Freescale Semiconductor, Ember Corporation, NEC, NXP, Renesas, Samsung, STMicroelectronics и Texas

Instruments. Провідні OEM-виробники – такі, як LG, Honeywell, Johnson Controls, Vantage/Legrand, Mitsubishi, Huawei, Motorola, NEC, Philips, Samsung, Eaton, Schneider Electric, Siemens, Yokogawa – розробляють своє устаткування на основі стандарту ZigBee.

Альянс успішно визнаний і отримує підтримку своїх рішень в області управління енергією і енергозбереження з боку комунальних промислових органів і співтовариств-прибічників побудови інтелектуальних мереж енергоспоживання. До Альянсу також долучилися і вже орієнтовані на рішення ZigBee декілька провідних світових компаній, що просувають "розумну енергетику", включаючи комунальних споживачів а також енергопостачальники.

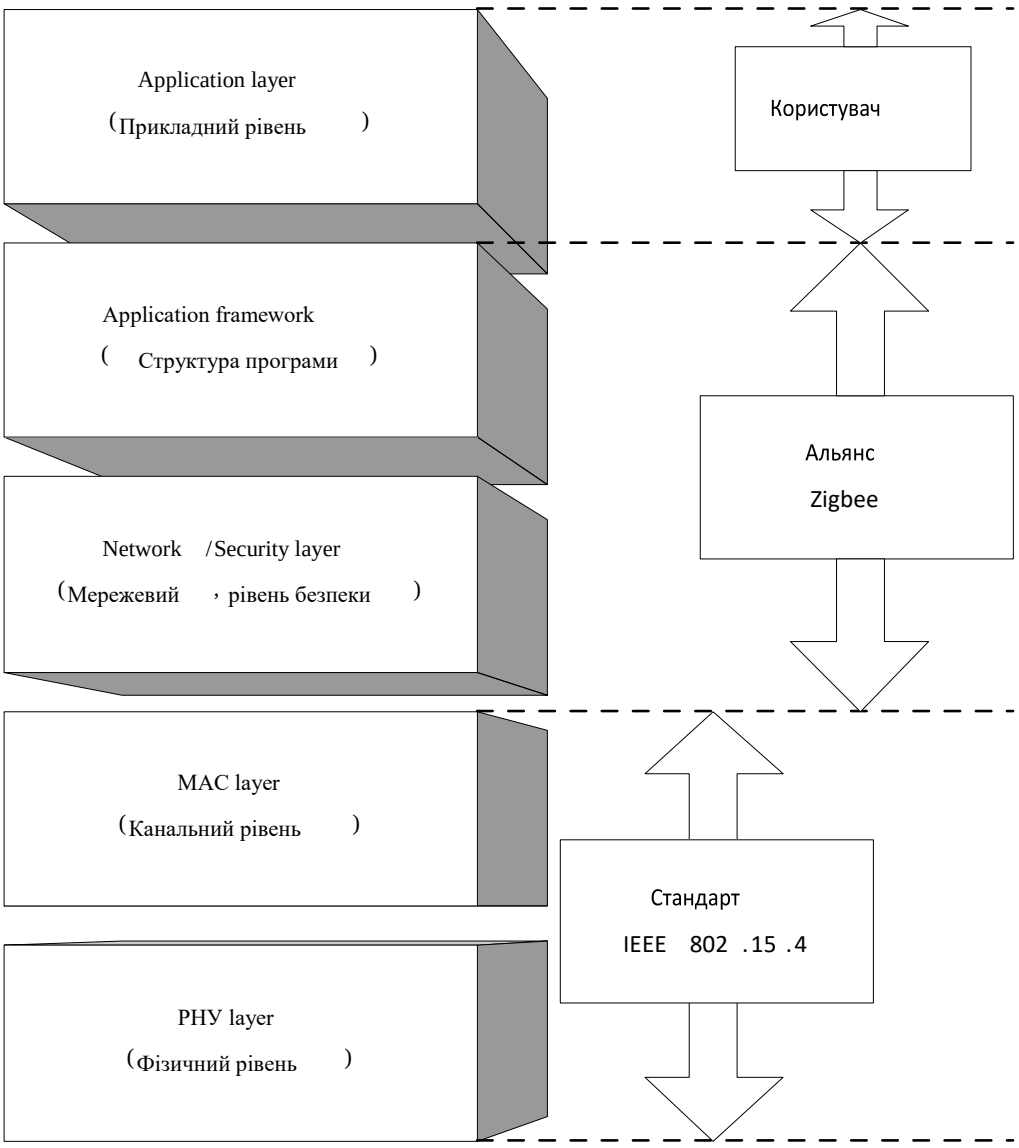

Рисунок 6 – Модель взаємодії стандарту IEEE 802.15.4 і технології бездротової передачі даних ZigBee у моделі OSI

З метою використання вже існуючих недорогих і легко встановлюваних рішень, а саме ZigBee-продуктів і послуг, такі системи як CenterPoint Energy, Sempra Utilities Southern California Edison і співпрацюють з такими компаніями-членами Альянсу: Siemens, Eaton, Cellnet, ItronSchneider Electric, Phillips, Johnson Controls, Control4, Legrand, Nivis, Golden Power, Control4, DCSI, Sensus Metering,

Nuri Telecom, Site Controls, Trilliant Networks, Silver Spring Networks, Talon Communications, Comverge, Tritech Technology і Viconics.

Використовуючи стандартом IEEE 802.15.4, призначеним для персональних мереж із низбкою швидкістю, Альянсу ZigBee у повній мірі вдалося використати відмостійкість, більший термін служби елементів живлення, і підтримку mesh-мереж, передбачених в стандарті, для того, щоб забезпечити відмостійкість та економність системи бездротового обміну даних з метою моніторингу і управління.

З метою побудови ZigBee, Альянс впровадив стандарти своїх специфікацій вище фізичного рівня (PHY) а також рівня керування доступу в середовищі передачі даних (MAC), що були прийняті в загальному стандарті IEEE 802.15.4, а також додавши рівні моделі взаємодії, до яких відносять мережевий рівень, прикладний рівень, рівень структури програми і рівень безпеки як показано на рисунку 1.5.

Мережевий рівень, протоколу бездротової передачі даних ZigBee, слугує для роспізнавання пристроїв і конфігурування мережі і підтримує три варіанти топології мережі, приведені на рисунку 7 . Хоча передавачі самі по собі недорогі, процес кваліфікування ZigBee включає в себе перевірку всіх вимог на фізичному рівні. На відміну від інших протоколів, які можуть мати поганий зв’язок або мати інші приховані проблеми, які виявляються у послабленні відгуку, ZigBee має визначені інженерні обмеження, які визначають енергоспоживання а також широти діапазону. Згідно з пунктом 6 стандарту 802.15.4-2006, передавачі мають пройти випробування на відповідність стандарту ISO 17025. Деякі виробники мають плани з’єднати мікроконтролер і радіо в одній мікросхемі.

Функціональна обробка основних даних, що утворюються сенсорами, відбувається на вузлі, або на досить потужниму комп'ютері. Оскільки перед тим як обробити дані, вони повинні бути отримані, вузол обов'язково оснащується антеною. Але у будь-якому випадку доступними для вузла можуть бути лише ті вузли, місцеположення яких є достатньо близьким від нього; іншими словами, вузол не може отримувати інформацію від кожного вузла безпосередньо. Проблеми отримання інформації від сенсорів, що збирається на вузлах, вирішується наступним чином.

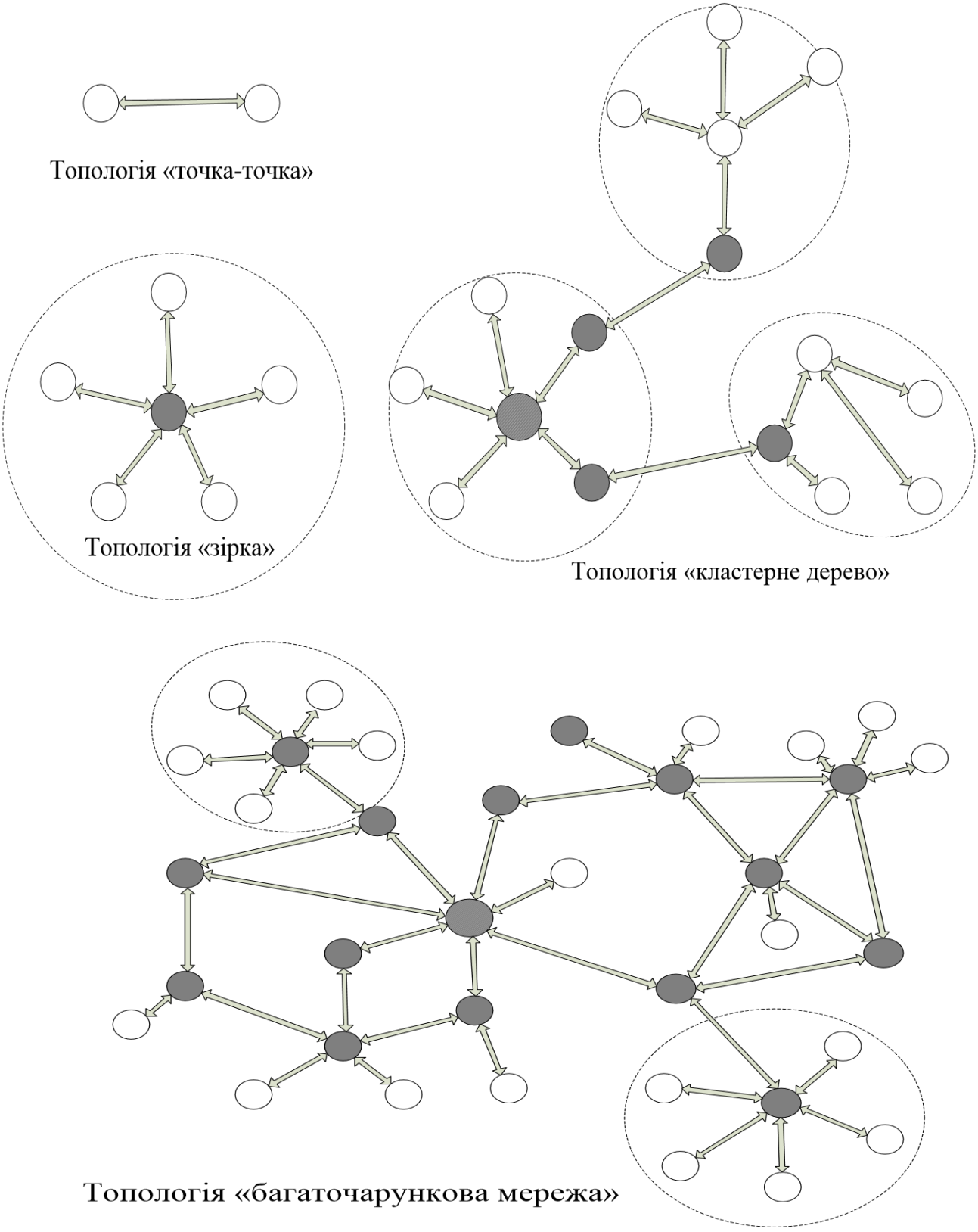

Рисунок 7 – Топології сенсорних мереж

Між вузлами інформація передається лише за допомогою приймачів, що працюють у певному діапазоні. Це – данні про стан пристроїв та інформація від сенсорів, що передається з датчиків та відомості про стан передачі даних. Інформація передається між певною кількістю вузлів що з’єднані між собою, і у результаті найближчі до шлюзу вузли передають йому усю інформацію. Якщо якась кількість вузлів перестає функціонувати коректно, робота сенсорної мережі після реконфігурації повинна продовжитись.

Але в такому випадку, природно, зменшується кількість джерел даних. Для виконання передачі і обробки даних на кожен вузол встановлюється спеціалізована операційна система. Операційна система для бездротових вузлів сенсорної мережі, як правило, менш складна, ніж загального призначення операційної системи, яка більше нагадує вбудовувану систему.

Зараз у багатьох бездротових сенсорних мережах встановлена TinyOS, яка розроблена в Університеті Берклі, вона класифікується як система з відкритим програмниме забезпечення з відкритим кодом; воно доступне за адресою: www.tinyos.net. TinyOS — це подійо-орієнтована операційна система реального часу, розрахована на роботу в умовах обмежених обчислювальних ресурсів.

Встановлення локалізації системи між вузлами є однією з необхідних умов, для створення працюючу БСМ. Проблема локалізації полягає у визначенні фізичного місця розташування (наприклад, широта, довгота, координати) об'єкта, який визначається. Мережа функціонує у вигляді орієнтованого графа, в якому кожна вершина — безпровідний вузол і кожній лінії між вузлами поставлено у відповідність певне значення — вірогідність помилки. Така задача є дуже важливою і актуальною, вона стосуються таких областей, як робототехніка, однорангових мереж, бездротових сенсорних мереж, мобільного зв'язку, військових систем, авіації та астрономії.

1.3. Локалізація об’єктів методом TDOA

TDOA (Time difference of arrival) - це метод, що дозволяє знайти місцеположення джерела сигналів за рахунок використання різниці в часі надходження звуку з джерела до сенсорів мережі. В якості вхідних даних приймається набір сигналів сенсорів і повертається масив координат джерел.

Відправлений джерелом імпульс буде отриманий двома розташованими на визначеній відстані один від одного сенсорами в різний час, це залежить від відстані між сенсором і джерелом. Різниця в часі між двома сенсорами порівнюється з гіперболоїдом, на якому знаходиться джерело. Розглянемо тепер третій приймач у третьому місці. Це забезпечить одне незалежне вимірювання TDOA (існує третя TDOA, але це залежить від перших двох TDOA і не надає додаткової інформації), а емітер розташований на кривій, визначений двома пересічними гіперболоїдами. Четвертий приймач потрібен для іншого незалежного TDOA, це дасть додатковий гіперболоїд, перетин кривої з допомогою цього гіперболоїда дає один або два розчини, а потім випромінювач знаходиться в одному або в одному з двох розчинів.. Проводиться ще один вимір різниці в часі приходу сигналу і отримання другого гіперболоїда, на якому знаходиться об'єкт. Перетин цих двох гіперболоїдів показує криву, по якій рухається об'єкт. Якщо є четвертий приймач, проводиться третій вимір різниці в часі, і перетин вже наявної кривої з третім гіперболоідом визначає конкретну точку в просторі. Таким чином, определяется точне місцезнаходження об'єкта. Принцип дії системи схематично зображено на рисунку 8.

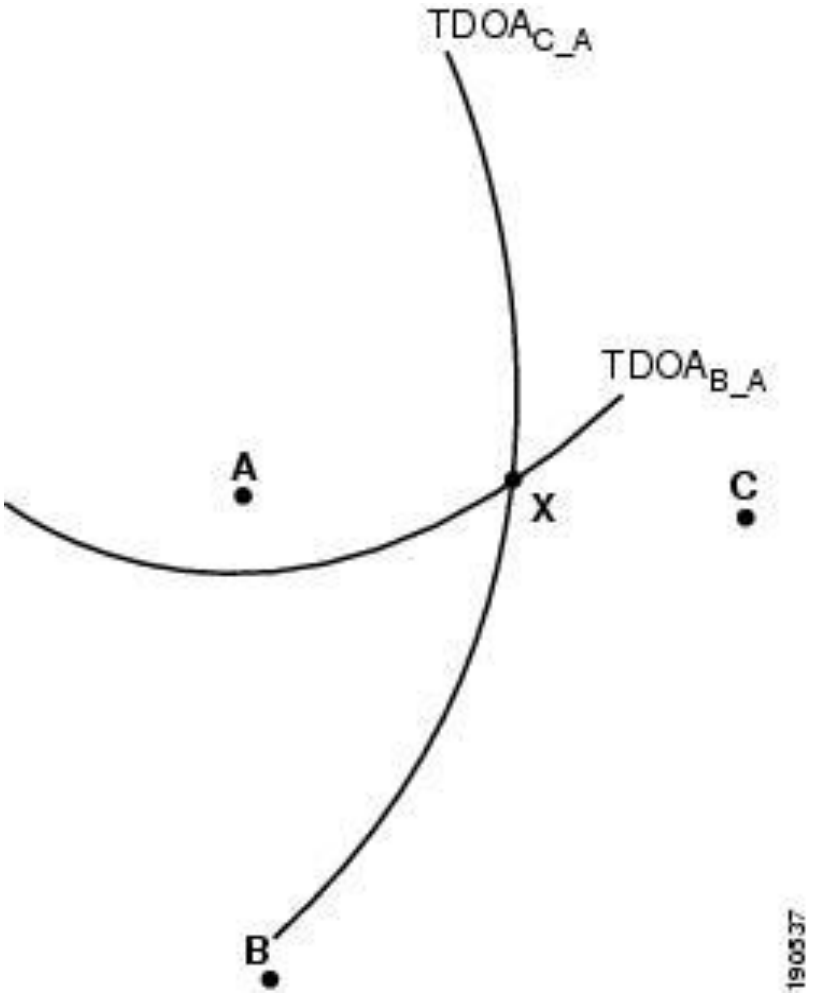

Рисунок 8 – Принцип метода TDOA

Визначення координат джерела зводяться до розв’язку матричного рівняння, де елементами шуканої матриці є координати джерела, а елементами відомих матриць є коефіцієнти, що залежать від координат мікрофонів та часу надходження сигналу на мікрофони.

Розглянемо джерело сигналу E на рисунку 9 у невідомому векторі розташування:

$$\vec{E} = (x, y, z) \tag{1}$$

Джерело знаходиться в межах діапазону N + 1 сенсорів координати яких відомі.

$$\vec{P}_0, \vec{P}_1, \cdots, \vec{P}_m, \cdots, \vec{P}_N. \tag{2}$$

Де

$$\vec{P}_m = (x_m, y_m, z_m)$$
$$0 \le m \le N \tag{3}$$

Відстань (Rm) від джерела до одного з сенсорів з точки зору координат становить:

$$R_m = \left|\vec{P}_m - \vec{E}\right| = \sqrt{(x_m - x)^2 + (y_m - y)^2 + (z_m - z)^2} \tag{4}$$

Візьмемо місцеположення сенсора (P0) за початок координат, тоді:

$$R_0 = \sqrt{x^2 + y^2 + z^2} \tag{5}$$

Виходячи з рівняння 4 маємо:

$$v\tau_m = vT_m - vT_0$$
$$v\tau_m = R_m - R_0 \tag{6}$$

Підвищення точності із збільшенням кількісті приймачів може стати проблемою для пристроїв з невеликими вбудованими процесорами через час, необхідний для вирішення кількох одночасних нелінійних рівнянь (4, 5 і 6). Вони можуть бути перетворені в систему лінійних рівнянь, коли є 3 або більше приймачів, що може зменшити час обчислення. Виходячи з рівняння 6:

$$R_m^2 = (v\tau_m + R_0)^2$$
$$R_m^2 = (v\tau_m)^2 + 2v\tau_m R_0 + R_0^2 \tag{7}$$

Отже:

$$0 = (v\tau_m)^2 + 2v\tau_m R_0 + R_0^2 - R_m^2$$
$$0 = (v\tau_m) + 2R_0 + \frac{(R_0^2 - R_m^2)}{v\tau_m}.$$

(8)

Видалення змінної 2R0 виключить всі квадратні кореневі змінні. Це робиться вирахуванням рівняння сенсора m = 1 від кожного іншого ($2 \le m \le N$):

$$0 = v\tau_m + 2R_0 + \frac{(R_0^2 - R_m^2)}{v\tau_m}$$
$$0 = -v\tau_1 - 2R_0 - \frac{(R_0^2 - R_1^2)}{v\tau_1}$$

$$0 = v\tau_m - v\tau_1 + \frac{(R_0^2 - R_m^2)}{v\tau_m} - \frac{(R_0^2 - R_1^2)}{v\tau_1}.$$

(9)

Виходячи з рівняння 4 та 5 маємо:

$$R_m^2 = x_m^2 + y_m^2 + z_m^2 - x\,2x_m - y\,2y_m - z\,2z_m + x^2 + y^2 + z^2$$
$$= x_m^2 + y_m^2 + z_m^2 - x\,2x_m - y\,2y_m - z\,2z_m + R_0^2$$

(10)

$$R_0^2 - R_m^2 = -x_m^2 - y_m^2 - z_m^2 + x\,2x_m + y\,2y_m + z\,2z_m$$

(11)

З'єднаємо рівняння 10 та 11 і запишемо як сукупність лінійних рівнянь розташування невідомого джерела E (x, y, z):

$$0 = xA_m + yB_m + zC_m + D_m$$

(11)

Де:

$$A_m = \frac{2x_m}{v\tau_m} - \frac{2x_1}{v\tau_1}$$
$$B_m = \frac{2y_m}{v\tau_m} - \frac{2y_1}{v\tau_1}$$
$$C_m = \frac{2z_m}{v\tau_m} - \frac{2z_1}{v\tau_1}$$
$$D_m = v\tau_m - v\tau_1 - \frac{x_m^2 + y_m^2 + z_m^2}{v\tau_m} + \frac{x_1^2 + y_1^2 + z_1^2}{v\tau_1}$$

(12)

Використовуючи рівняння 12 для обчислення чотирьох констант від вимірюваних відстаней і часу для кожного приймача $2 \leq m \leq N$ отримаємо набір N-1 неоднорідних лінійних рівнянь. Існує багато надійних методів лінійної алгебри, які використати для обчислень значень (x, y, z), які і будуть координатами місцеположення джерела сигналу E.

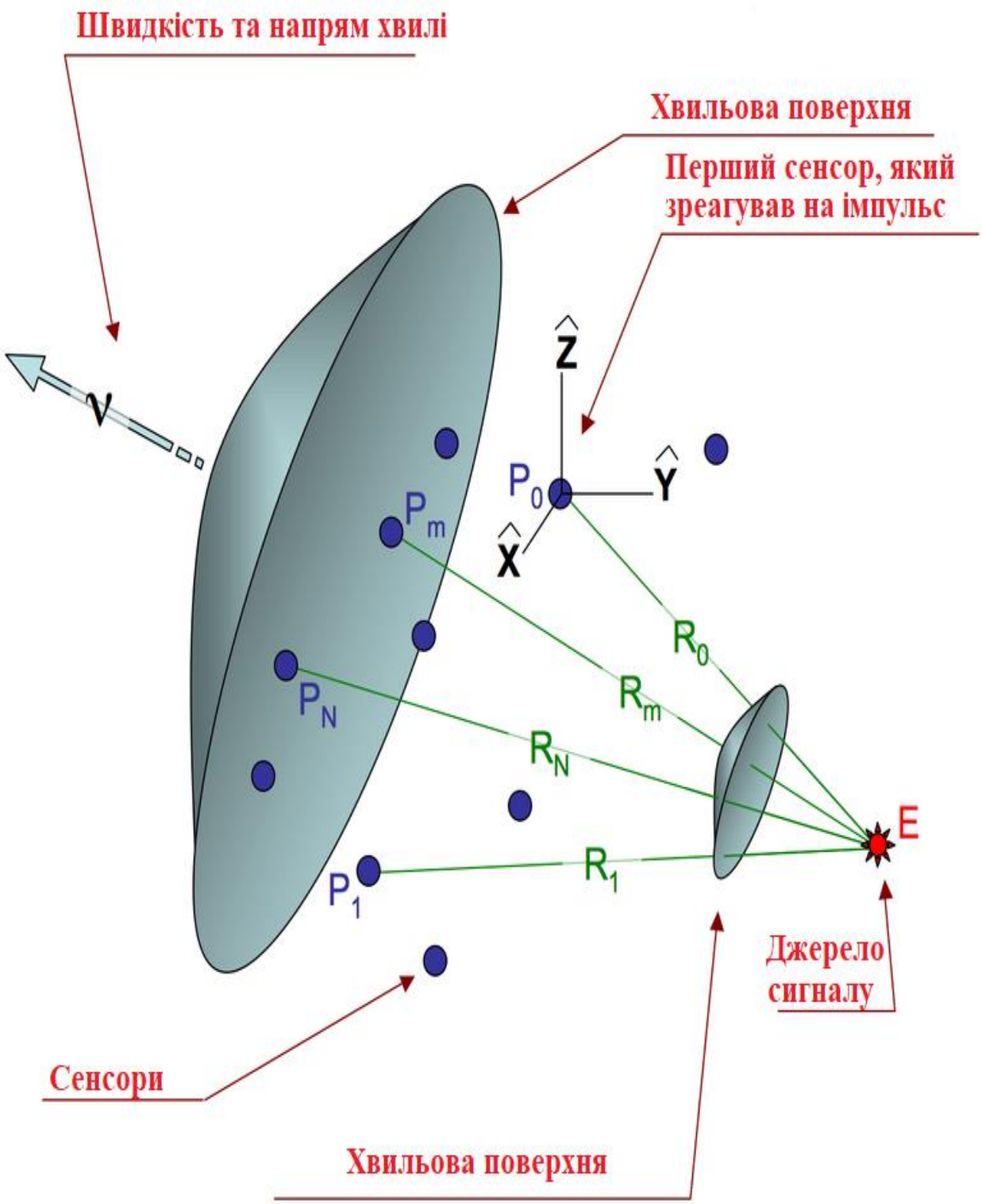

Рисунок 9 – Схема локалізації методом TDOA

На рисунку 10, 11 та 12 зображено графіки різниці часу надходження імпульсного, широкосмугового та вузькосмуговий сигналу на сенсори зображені на рисунку 9.

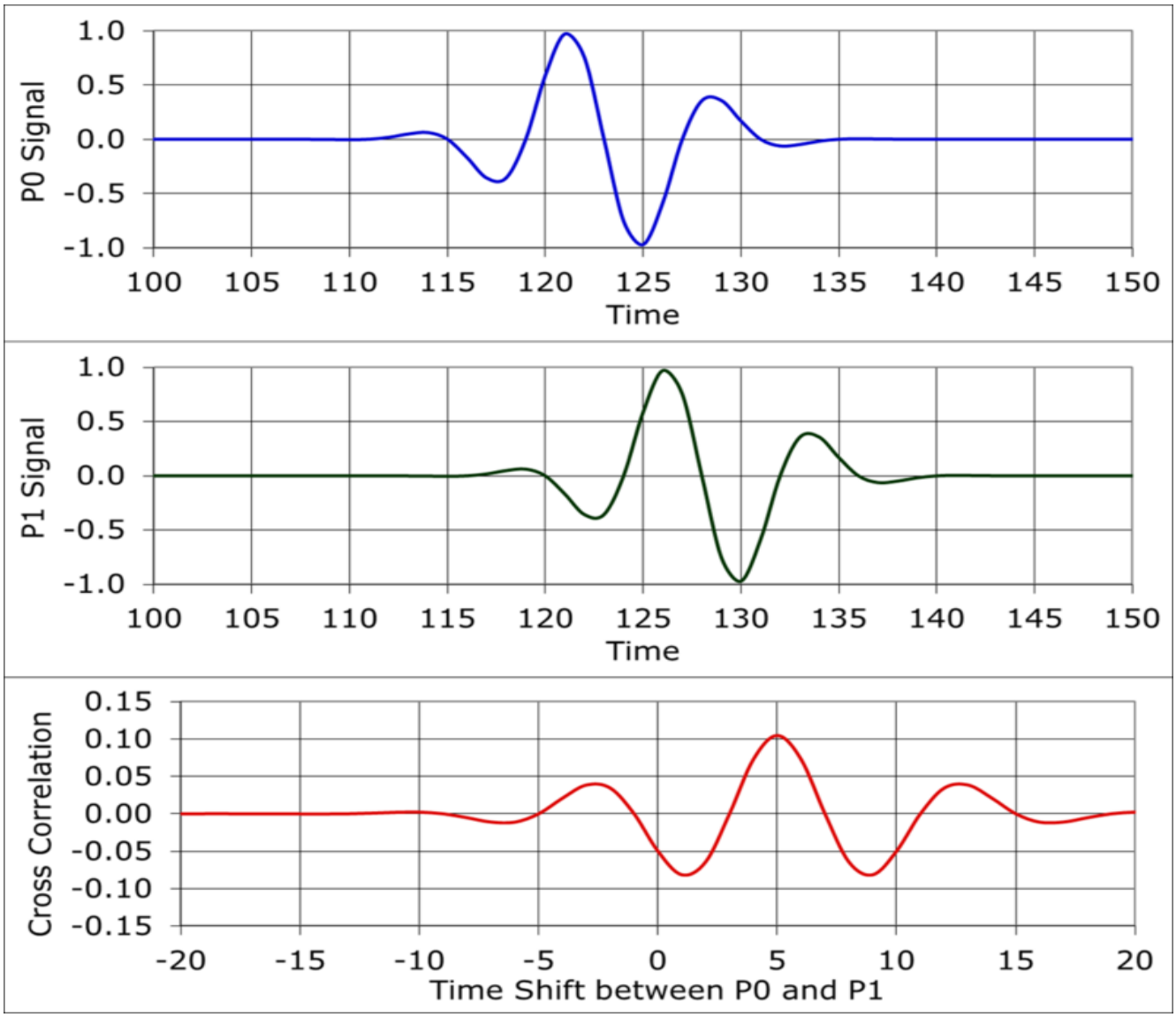

Рисунок 10 – Різниця часу надходження система імпульсного сигналу на сенсори P0 та P1

Імпульсний сигнал – це сигнал із значним збільшенням інтенсивності й наступним різким зменщенням інтенсивності до вихідної величини. Інше визначення: коротка зміна амплітуди на протягом незмінного несучого сигналу.

Можливе періодичне повторення таких змін амплітуди. Імпульс характеризується тривалістю, значною крутизною діапазону збільшення й крутизною діапазону зменшення.

Під час певної кількості імпульсів відрізок часу між імпульсами більший, ніж тривалість змін потужності самого імпульсу.

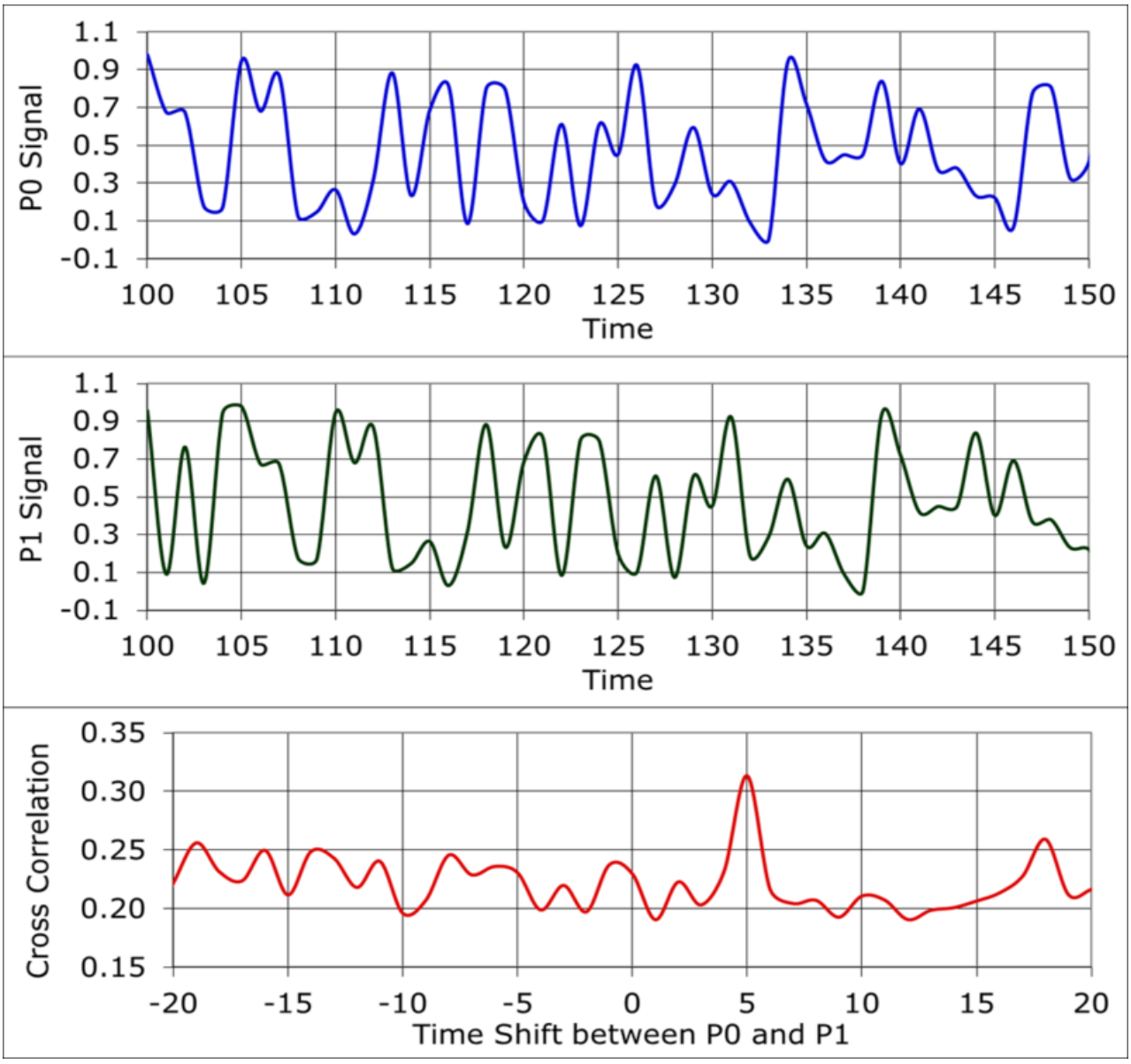

Рисунок 11 – Різниця часу надходження система широкосмугового сигналу на сенсори P0 та P1

Широкосмуговий сигнал – це сигнал, ширина діапазону якиго порівнянна з центральною частотою.

Часто використовується коефіцієнт «1/10», тобто якщо ширина діапазону зміни частот становить близько 1/10 від частоти, на якій приймається сигнал, то сигнал вважається широкосмуговим. При більш вузькому діапазоні сигнал буде вузькосмуговим, при більш широкому - надширокосмугові.

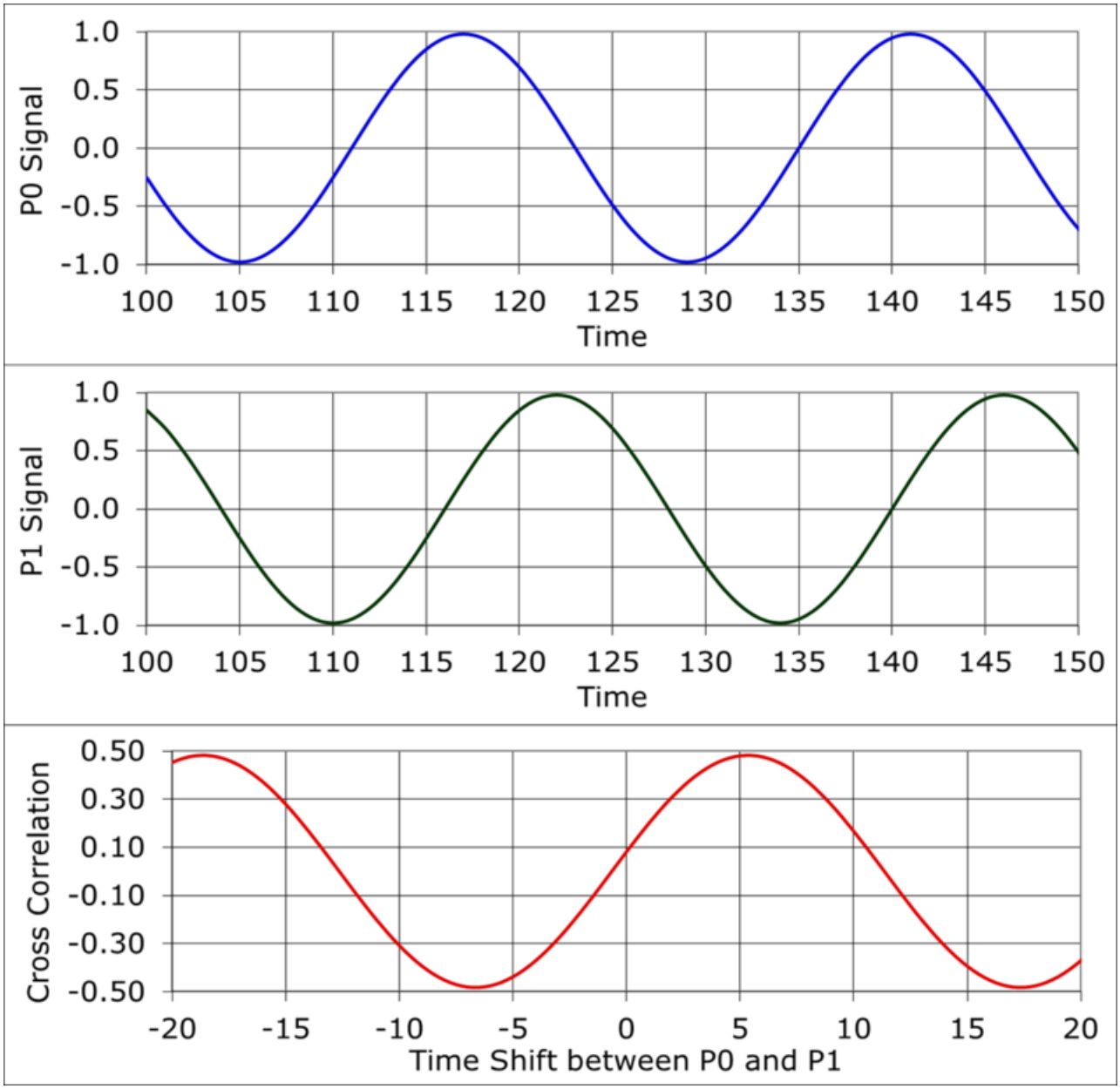

Рисунок 12 – Різниця часу надходження система вузькосмугового сигналу на сенсори P0 та P1

Вузькосмуговим (відносно вузькосмуговими) називається сигнал, який знаходиться в діапазоні, ширина якого порівняно менша ніж середня частота. Вузькосмугові сигнали часто використовуються системах передачі даних з частотним розподіленням (в радіоканалах і в багатоканальних системах). Також сигнал є вузькосмуговим, якщо відношення максимальної частоти діапазону до мінімальної не більше двох, що не суперечить попередньому визначенню.

1.4. Цілі та вимоги розподілених систем обробки даних

Розподілена система - це така система, в якій взаємодія і синхронізація програмних компонентів, які виконуються на незалежних мережевих комп'ютерах, здійснюється за допомогою передачі повідомлень.

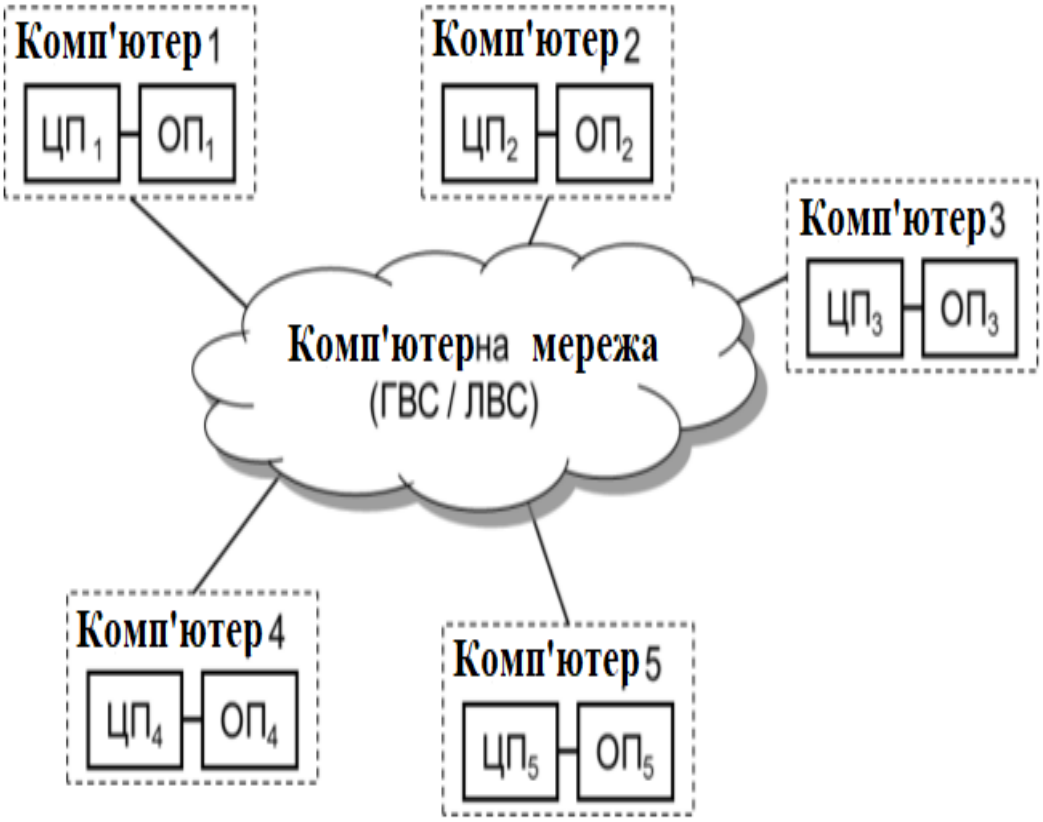

Рисунок 13 – Розподілена система обробки даних

ЦП – центральний процесор; ОП – операційна пам’ять

Розподілені системи зазвичай мають такі ознаки:

- Відсутність однакового відліку часу у компонентах розподіленої системи. Це важливе обмеження для вирішення задач проектування і побудови розподілених систем.

Таку поведінку визначає територіальний розподіл компонентів системи, а саме процесорів, які можуть входти до складу системи, але що більш важливо, з нього виходить, що існує відсутність синхронності в їх роботі.

- Відсутність спільної пам'яті в системі. Це важлива проблема, з якої виникає необхідність передачі повідомлень між програмними компонентами розподіленої системи з метою їх взаємодії і синхронізації. Крім того, ця характеристика має на увазі відсутність єдиного для всіх процесорів фізичного часу. Тут варто зазначити, що деякі розподілені системи надають своїм користувачам абстракцію єдиного адресного простору для всіх процесорів за допомогою механізмів розподіленої пам'яті, що (англ. Distributed Shared Memory, DSM).

  В такому випадку, не розглядаючи складності конкурентного доступу декількох процесорів до одного й того ж сегменту пам'яті, для кожного процесора розподілену пам'ять, що розділяється слід представляти як цілком нормальну організацію віртуальної пам'яті, в якій не використовується власний диск у якості тимчасового сховища інформації, а використовується оперативна пам'ять віддаленого комп'ютера. Виходячи з цього у літературі по розподіленим системам зазвичай, серед іншого, додатково розглядаються певні аспекти організації спільної пам'яті в багатопроцесорних системах.
- Географічний розподіл. Можна зазначити, що із збільшенням віддаленості процесорів один від одного фізично, тим зрозуміліше, що система буде вважатись розподіленою. Однак немає необхідності у тому, щоб комп'ютери були з’єднані в глобальну обчислювальну мережу (ГОС). Останнім часом кластер із звичайних робочих станцій (англ. Cluster Of Workstation, COW), з'єднаних за допомогою локальної обчислювальної мережі (ЛОМ), також все частіше розглядається як невелика розподілена система. При цьому все

обладнання такої розподіленої системи може знаходитися в одному або декількох сусідніх будинках. Подібні кластери COW стають дедалі популярнішими через відносно низьку вартість що в неї входять з одного боку і непоганий продуктивності - з іншого. Наприклад, ядро пошукової системи компанії Google побудовано за архітектурою COW.

- Незалежність і гетерогенність. Комп'ютери, що входять до складу розподіленої системи слабо пов'язані - вони можуть мати різний склад і різну продуктивність і, отже, забезпечувати різний час виконання ідентичних задач. Зазвичай вони не є частинами однієї спеціалізованої системи, але функціонують спільно, надаючи свої служби один одному для виконання спільного завдання. Більш того, в загальному випадку на комп'ютерах, що становлять розподілену систему, можуть виконуватися різні операційні системи.

Для обробки даних сенсорів доцільно використати розподілену систему серверів. Серед основних цілей організації такої системи можна виділити наступні.

Географічно розподілене обчислювальне середовище. Сьогодні в багатьох випадках саме обчислювальне середовище за своєю природою являє собою територіально розподілену систему. І збільшення обчислювального ресурсу не потребує територіальної прив’язки.

Вимога збільшення продуктивності обчислень. Швидкодія традиційних однопроцесорних систем стрімко наближається до своєї межі. Різні архітектури (такі як суперскалярна архітектура, матричні і векторні процесори, однокристальних багатопроцесорні системи) покликані збільшувати продуктивність обчислювальних систем за рахунок різних механізмів паралельного виконання команд. Однак всі ці прийоми здатні підвищити продуктивність максимум в десятки разів по порівняння з класичними

повартоовними рішеннями. Крім того, масштабованість подібних підходів залишає бажати кращого. Щоб підвищити продуктивність в сотні або тисячі разів і при цьому забезпечувати хорошу масштабованість рішення необхідно звести воєдино численні процесори і забезпечити їх ефективну взаємодію. Цей принцип реалізується у вигляді великих багатопроцесорних систем і багатомашинних комплексів.

Спільне використання ресурсів. Важливою метою створення і використання розподілених систем є надання користувачам (і додаткам) доступу до віддалених ресурсів і забезпечення їх спільного використання. У цьому формулюванні термін ресурс відноситься як до компонентів апаратного забезпечення обчислювальної системи, так і до програмних абстракцій, з якими працює розподілена система. Наприклад, користувач комп'ютера 1 може використовувати дисковий простір комп'ютера 2 для зберігання своїх файлів. Або додаток А може використовувати вільну обчислювальну потужність декількох комп'ютерів для прискорення власних розрахунків. Розподілені бази даних і розподілені системи об'єктів можуть бути відмінним прикладом спільного використання програмних компонентів, коли відповідні програмні абстракції розподілені по декількох комп'ютерів і узгоджено обслуговуються декількома процесами, що утворюють розподілену систему.

Відмовостійкість. У традиційних нерозподілених обчислювальних системах, побудованих на базі одиничного комп'ютера (можливо високопродуктивного), вихід з ладу одного з його компонентів зазвичай призводить до непрацездатності всієї системи. Такий збій в одному або декількох компонентах системи називають частковим відмовою, якщо він не зачіпає інші компоненти. Характерною рисою розподілених систем, яка відрізняє їх від одиничних комп'ютерів, є стійкість до часткових відмов, тобто система продовжує функціонувати після часткових відмов, правда, трохи знижуючи при цьому загальну продуктивність. Подібна можливість досягається за рахунок надмірності, коли в систему додається додаткове

обладнання (апаратна надмірність) або процеси (програмна надмірність), які уможливлюють правильне функціонування системи при непрацездатності або некоректної роботи деяких з її компонентів. В цьому випадку розподілена система намагається приховувати факти відмов або помилок в одних процесах від інших процесів. Наприклад, в системах з потрійним модульним резервуванням (англ. Triple Modular Redundancy, TMR) використовуються три однакових обчислювальних модуля, що здійснюють ідентичні обчислення, а коректний результат визначається простим голосуванням.

Ефективна розподілена система повинна відповідати таким вимогам: прозорість (англ. Transparency), відкритість (англ. Openness), безпека (англ. Security), масштабування (англ. Scalability). Однак варто зазначити, що, незважаючи на уявну простоту і очевидність зазначених вимог, їх реалізація на практиці є непростим завданням.

Під прозорою розподіленою системою розуміють таку систему, яка здатна приховувати свою розподілену природу, тобто, розподіл процесів і ресурсів по визначеній кількості комп'ютерів, і являється для користувачів і розробників додатків у вигляді єдиної централізованої комп'ютерної системи. Стандарти еталонної моделі для розподіленої обробки у відкритих системах Reference Model for Open Distributed Processing (RM-ODP) визначають декілька типів прозорості. Найбільш важливі з них перераховані нижче.

• Прозорість доступу (англ. Access transparency). Незалежно від способів доступу до ресурсів і їх внутрішнього подання, звернення до локальних і віддалених ресурсів здійснюється однаковим чином. На базовому рівні ховається різниця архітектур обчислювальних платформ, але, що більш важливо, досягається угода про те, як ресурси різнорідних машин, будуть представлятися користувачам розподіленої системи єдиним чином. Як приклад можна привести прикладний

програмний інтерфейс (англ. Application programming interface, API) для роботи з файлами, що зберігаються на безлічі комп'ютерів різних архітектур, який надає однакові виклики операцій як з локальними, так і з віддаленими файлами.

• Прозорість розташування (англ. Location transparency). Дозволяє звертатися до ресурсів без знання їх фізичного розташування. У цьому випадку ім'я запитуваного ресурсу не повинно давати жодного уявлення про те, де ресурс розташований. Тому важливу роль для забезпечення прозорості розташування грає іменування ресурсів. Наприклад, щоб відправити повідомлення електронної пошти на адресу user@company.com не потрібно знати фізичного місця розташування одержувача, його поштової скриньки або поштового сервера. У свою чергу звернення до файлу \\ server \ foo має на увазі знання імені сервера, на якому він розташований, а значить, не є повністю прозорим з точки зору розташування.

• Прозорість переміщення (англ. Migration transparency). Переміщення ресурсу або процесу в інше фізичне місце розташування залишається непомітним для користувача розподіленої системи. Тут варто зазначити, що виконання вимоги прозорості місцеположення не гарантує прозорості переміщення. Іншими словами, якщо розподілена система приховує місце розташування ресурсу, це не означає, що його можна змінити непомітно для користувача.

• Розподілені файлові системи дозволяють монтувати файлові системи віддалених комп'ютерів в локальний простір імен клієнта, надаючи єдине дерево каталогів і тим самим забезпечуючи прозорість розташування. Однак якщо файли на віддалених комп'ютерах будуть переміщені в інше місце, в більшій частині розподілених файлових систем вони стануть недоступні для користувача.

• Прозорість зміни місця розташування (англ. Relocation transparency). Більш суворе по відношенню до попереднього вимога приховати факт переміщення ресурсу під час його використання. Прикладом можуть служити мобільні

користувачі, використовують стільникові телефони. В цьому випадку, якщо розглядати абонента в якості користувача розподіленої системи, а викликається - як її ресурсу, то система буде прозорою з точки зору зміни місця розташування. Дійсно, переміщення "ресурсу" зі стільника в стільник в процесі розмови залишається непомітним для абонента.

• Прозорість реплікації (англ. Replication transparency). Якщо для підвищення доступності або збільшення продуктивності використовується кілька копій ресурсу (реплік), цей факт залишається прихованим від користувача, і він вважає, що в системі присутній тільки один екземпляр ресурсу.

• Забезпечення прозорості реплікації необхідно, щоб всі репліки мали одне і те ж ім'я, очевидно, що не залежить від місця розташування копії ресурсу. Таким чином, системи, які забезпечують прозорість реплікації, також повинні підтримувати і прозорість розташування.

• Прозорість одночасного доступу (англ. Concurrency transparency). Дозволяє декільком користувачам (конкуруючим процесам) одночасно виконувати операції над загальним, спільно використовуваних ресурсом без взаємного впливу один на одного. Інакше кажучи, ховається факт використання ресурсу іншими користувачами (процесами). Варто відзначити, що сам ресурс повинен залишатися в несуперечливому стані, що може досягатися, наприклад, за допомогою механізму блокувань, коли користувачі (процеси) по черзі отримують виключні права на запитуваний ресурс.

• Прозорість відмов (англ. Failure transparency). Мається на увазі, що система обов’язково має приховувати часткові відмови, дозволяючи при цьому користувачам і додаткам виконувати свою роботу незалежно від помилок в апаратному чи програмному компоненті розподіленої системи, а також приховати факт їх майбутнього відновлення. У зв'язку з тим, що будь-який процес, комп'ютер

або мережеве з'єднання можуть відмовляти незалежно від інших в довільні моменти часу, кожен компонент розподіленої системи повинен бути готовий до збоїв в інших компонентах і обробляти подібні ситуації відповідним чином.

Ступінь прозорості. Важливо відзначити, що ступінь, до якої кожен із перелічених вище властивостей має бути виконано, може сильно варіюватися в залежності від задач побудови розподіленої системи. Дійсно, повністю приховати розподіл процесів і ресурсів навряд чи вдасться. Через обмеження в швидкості передачі сигналу, затримка на звернення до ресурсів, територіально віддалених від клієнта, завжди буде більше, ніж до ресурсів, розташованим поблизу.

Згідно з визначенням, прийнятим комітетом IEEE POSIX 1003.0, відкрита система - це система, яка реалізує відкриті специфікації (стандарти) на інтерфейси, служби і підтримувані формати даних, достатні для того, щоб забезпечити:

• можливість перенесення розробленого прикладного програмного забезпечення на широкий діапазон систем з мінімальними змінами (мобільність додатків, переносимість);

• спільну роботу (взаємодія) з іншими прикладними програмами на локальних і віддалених платформах (інтероперабельність, здатність до взаємодії);

• взаємодія з користувачами в стилі, що полегшує останнім перехід від системи до системи (мобільність користувача).

Ключовий момент у цьому визначенні - використання поняття відкрита специфікація, яке, в свою чергу, визначається як загальнодоступна специфікація, яка підтримується відкритим, гласним погоджувальною процесом, спрямованим на постійну адаптацію до нових технологій, і відповідає стандартам.

Згідно з цим визначенням відкрита специфікація не залежить від конкретної технології, тобто не залежить від конкретних технічних і програмних засобів або

продуктів окремих виробників. Відкрита специфікація однаково доступна будь-якій зацікавленій стороні. Більш того, відкрита специфікація знаходиться під контролем громадської думки, тому зацікавлені сторони можуть брати участь в її розвитку.

У загальному випадку масштабованість визначають, як здатність обчислювальної системи ефективно справлятися зі збільшенням числа користувачів або підтримуваних ресурсів без втрати продуктивності і без збільшення адміністративного навантаження на її управління. При цьому систему називають масштабується, якщо вона здатна збільшувати свою продуктивність при додаванні нових апаратних засобів. Іншими словами, під масштабністю розуміють здатність системи рости разом з ростом навантаження на неї.

Для розподілених систем зазвичай виділяють кілька параметрів, що характеризують їх масштабованість: кількість користувачів і кількість компонентів, що складають систему, ступінь територіальної віддаленості мережевих комп'ютерів системи один від одного і кількість адміністративних організацій, які обслуговують частини розподіленої системи. Тому масштабованість розподілених систем також визначають за відповідними напрямами:

• Здатність навантаження масштабованість. Здатність системи збільшувати свою продуктивність при збільшенні навантаження шляхом заміни існуючих апаратних компонентів на більш потужні або шляхом додавання нових апаратних засобів. При цьому перший випадок збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності називають вертикальним масштабуванням, а другий, що виражається в збільшенні кількості мережевих комп'ютерів (серверів) розподіленої системи - горизонтальним масштабуванням.

• Географічна масштабованість. Здатність системи зберігати свої основні характеристики, такі як продуктивність, простота і зручність використання, при територіальному рознесення її компонентів від більш локального взаємного розташування до більш розподіленого.

• Адміністративна масштабованість. Характеризує простоту управління системою при збільшенні кількості адміністративно незалежних організацій, що обслуговують частини однієї розподіленої системи.

1.5 Аналіз програмних платформ для побудови розподілених обчислень.

Ефективна розподілена обробка великих обсягів даних є нетривіальним завданням, для вирішення якої в 2004 році компанія Google розробила модель розподілених обчислень під назвою MapReduce. MapReduce – фреймворк, що надає спеціальну програмну модель і систему для обробки та генерації великої кількості даних, яка може бути застосована для багатьох задач, що виникають в реальному житті. Програми, написані для цієї системи, автоматично розподіляються і виконуються на легко масштабованому кластері. Система також обробляє збої машин в кластері, координує сполучення між комп'ютерами усередині кластера, мінімізуючи навантаження на мережу і сховище даних. У зв'язку з тим, що реалізація Google є закритою, інші великі компанії, такі як Yahoo !, Facebook і Amazon активно розробляють свої реалізації моделі MapReduce або покращують вже існуючі.

MapReduce використовуються для обробки великої кількості даних, і тому зазвичай запускаються на кластерах з декількох вузлів. В основі моделі лежать дві функції Map і Reducев цілому аналогічні тим, які часто використовуються в функціональному програмуванні. Всі MapReduce програми складаються з двох стадій:

- На стадії Map вхідні дані поділяються на частини і розсилаються на вузли. На кожному вузлі викликається функція Map, результатом якої є список з пар: (проміжний ключ, проміжне значення). Далі, отримані дані за допомогою функції partition розподіляються по машинах, які будуть виконувати подальшу обробку. Трохи докладніше зупинимося на функції partition, так як вона буде багато в чому ключовий для даної роботи. Partition є відображенням безліч проміжних ключів у безліч [0, k - 1], де k - кількість машин. У класичному варіанті MapReduce однакові проміжні ключі повинні потрапити на одну і ту ж машину для коректної обробки, тому ця функція не повинна мати побічних ефектів.
- На стадії Reduceпроміжні дані на кожній машині упорядковано і групуються по проміжному ключу, після чого на них викликається функція згортки (reduce). Функція згортки бере на вхід проміжний ключ і список всіх проміжних значень, зіставлених йому, і повертає деяку пару.

Перевага такої моделі в тому, що стадії Map і Reduce можна виробляти паралельно на певній кількості машин, що відсутні довільно розподілені дані і, як навартоок, постійна взаємодія з мережею, необхідна для синхронізації даних на вузлах. Все це дозволяє системі бути надійною і масштабованою. На рисунку 14 відображені основні стадії MapReduce програм:

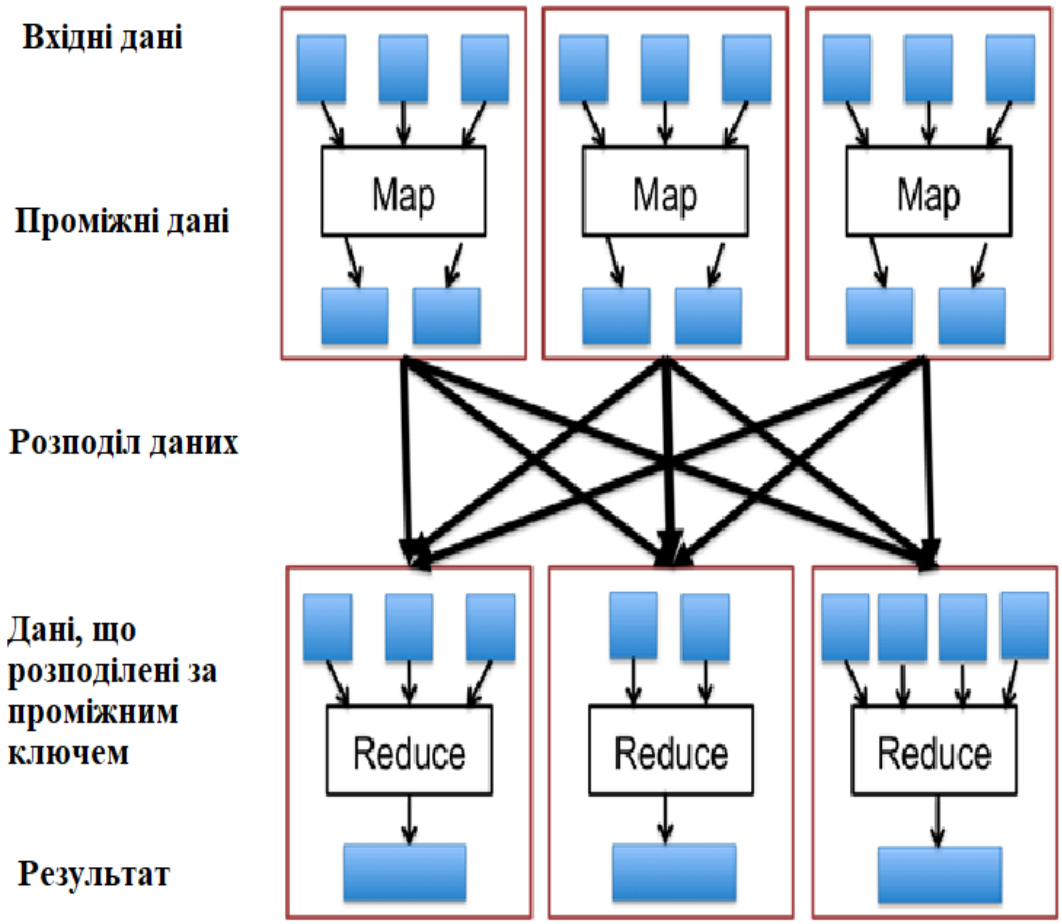

Рисунок 14 – Модель MapReduce

У платформі Hadoop MapReduce підтримується можливість читання вхідних даних в декількох різних форматах. Наприклад, в режимі "text" кожен рядок трактується як пара "ключ-значення", де ключ - це зміщення до цього рядка від початку файлу, а значення - вміст рядка.

В іншому поширеному форматі вхідні дані подаються у вигляді пар "ключ-значення", відсортованих за значеннями ключа. У кожній реалізації формату вхідних даних відомо, яким чином варто расщеплять дані на осмислені частини, які обробляються окремими завданнями Map (наприклад, дані формату "text" розщеплюються тільки по межами рядків).

Користувачі можуть додати до реалізації власні формати вхідних даних, забезпечивши нову реалізацію інтерфейсу reader (в реалізації Hadoop - RecordReader). Reader не обов'язково повинен читати дані з файлу, можна легко визначити reader, який читає дані з бази даних або з деякою структури в віртуальної пам'яті. Аналогічним чином, підтримуються можливості генерації даних в різних форматах, і є проста можливість визначення нових форматів результуючих даних.

MapReduce забезпечують масштабованість технології до десятків тисяч вузлів, її відмовостійкість, дешевизну завантаження даних і можливість використання явно написаного коду, який добре распаралелюється.

Альтернативною вільно доступною реалізацією MapReduce (з відкритими вихідним кодом) є проект Hadoop спільноти APACHE. Вона заснована на використанні розподіленої файлової системи HDFS (Hadoop Distributed File System), також розробленої в проекті Hadoop. Реальну популярність MapReduce принесла саме реалізація Hadoop в силу своєї доступності і відкритості, а широке використання Hadoop MapReduce в різних довартоницьких і довартоницьких проектах приносить безсумнівну користь цієї системі, стимулюючи розробників до її постійного вдосконалення.

Основними компонентами (ядро) «Hadoop» є:

- «Hadoop Distributed File System» (HDFS) – розподілена файлова система, що дозволяє зберігати інформацію практично необмеженого обсягу.

- «Yarn» - програмна модель для управління ресурсами і менеджменту задач, в тому числі включає програмну модель «MapReduce» (використовуються в нових версіях «Hadoop» замість «MapReduce»).

- «Hadoop common» - бібліотеки і утиліти, які використовуються іншими модулями «Hadoop».

- «Hadoop MapReduce» - програмний каркас для програмування розподілених обчислень в рамках парадигми «MapReduce».

На рисунку 15 показана схема блоків APACHE Hadoop.

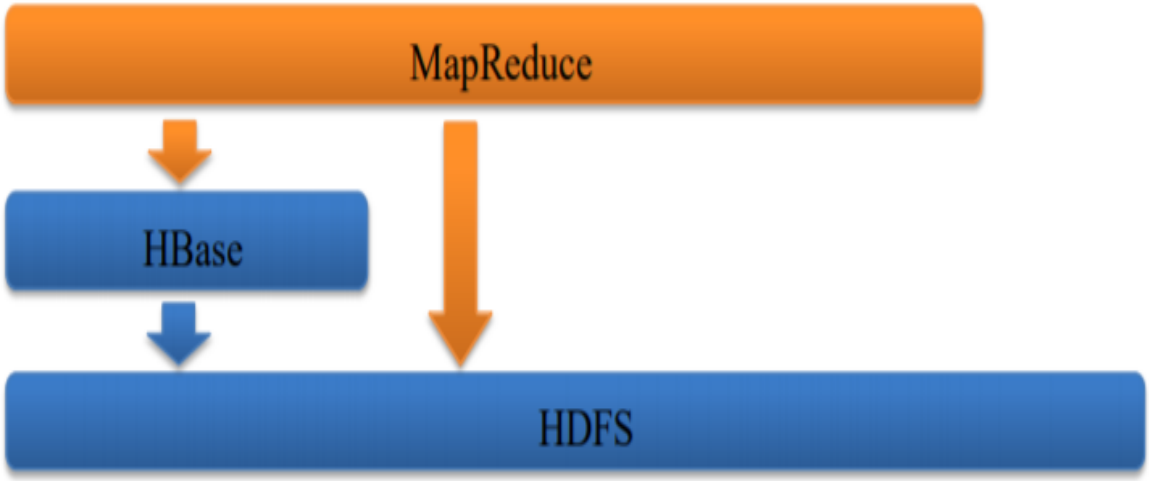

Рисунок 15 – схема основних компонентів екосистеми «Hadoop»

Hadoop MapReduce спирається на розподілену файлову систему HDFS (Hadoop Distributed File System). Файли HDFS мають блочну структуру, і блоки одного файлу розподіляються по вузлах даних (DataNode). Файлова система працює під централізованим управлінням виділеного вузла імен (NameNode), в якому підтримуються метадані про файли (в тому числі, про їх розмірах, про розміщення блоків і їх реплік і т.д.). У самому середовищі Hadoop MapReduce відповідно до підтримуються один вузол-розпорядник (в Hadoop він називається JobTracker) і багато узловісполнітелей (тут TaskTracker). У вузлі JobTracker планується виконання MR-задач, а також відвартоковуються дані про завантаження вузлів TaskTracker і доступних ресурсах. Кожне завдання розбивається на завдання Map і Reduce, які призначаються вузлом JobTracker вузлів TaskTracker з

урахуванням вимог локальності даних і балансування навантаження. Вимога локальності задовольняється за рахунок того, що JobTracker намагається призначати кожну задачу Map тому вузлу TaskTracker, для якого дані, що обробляються цим завданням, є локальними.

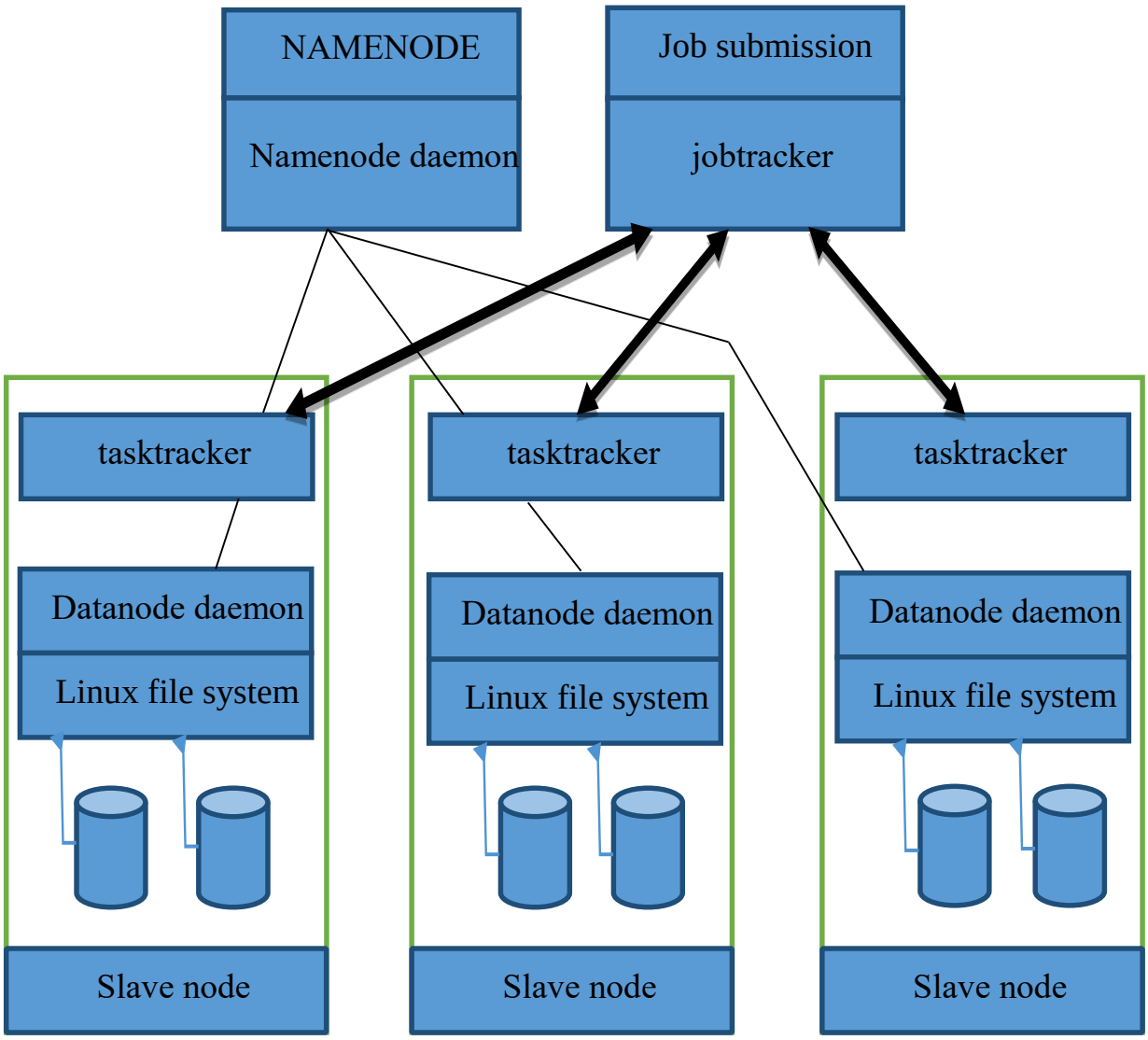

Рисунок 16 – Архітектура повного кластера Hadoop, який складається з трьох окремих компонентів: майстер HDFS (так званий NameNode), вузол

уявлення завдання (так званий JobTracker), і багато підлеглих вузлів (три показано тут).

Кожен з підлеглих вузлів запускає TaskTracker для виконання Map і Reducesавдання і DataNode для обслуговування HDFS даних Hadoop кластерної архітектури.

Балансування навантаження досягається шляхом призначення задач всім доступним вузлам TaskTracker. Вузли TaskTracker періодично посилають в вузол JobTracker контрольні повідомлення з інформацією про свій стан. Для забезпечення доступу до вхідних даних MR-завдання підтримується бібліотека InputFormat. У Hadoop MapReduce є кілька реалізацій цієї бібліотеки, одна з яких дозволяє всім завданням одного MR-завдання звертатися до JDBC-сумісної базі даних.

Основні переваги APACHE MapReduce:

- Ефективна робота з великим об'ємом даних.

- Масштабованість.

- Відмовостійкість.

- Відкритий вихідний код.

Недоліки APACHE MapReduce:

- Відсутність автоматичного запуску головного сервера в разі його збою (дана функціональність реалізована в GFS).

- «NameNode» - є найслабшою ланкою в системі. У випадку його збою, впаде все система.

- Немає багатопоточного запису. Один процес запису на файл. Дані дописуються в кінець файлу.

- Не підходить для великої кількості маленьких файлів.

- Ефективність застосування MapReduce знижується при малій кількості комп'ютерів.

- Не можна визначити закінчення стадії Map.

- Затримки у виконанні будь-якої запущеної Map завдання веде до затримки виконання завдання цілком.

- Збій вузла JobTracker призводить до простою всього кластера.

- Для запису проміжних даних використовується дисковий простір вузлів кластера, що значно вповільнює систему.

APACHE SPARK - фреймворк з відкритим вихідним кодом для реалізації розподіленої обробки неструктурованих і слабоструктурованих даних. На відміну від класичного обробника з ядра Hadoop, що реалізує дворівневу концепцію MapReduce з дисковим сховищем, використовує спеціалізовані примітиви для рекуррентної обробки в оперативній пам'яті, завдяки чому дозволяє отримувати значний виграш в швидкості роботи, зокрема, можливість багаторазового доступу до завантажених в пам'ять призначених для користувача даних робить бібліотеку привабливою для SPARK реалізований на мові Scala. SPARK використовує мову Scala в якості середовище розробки.

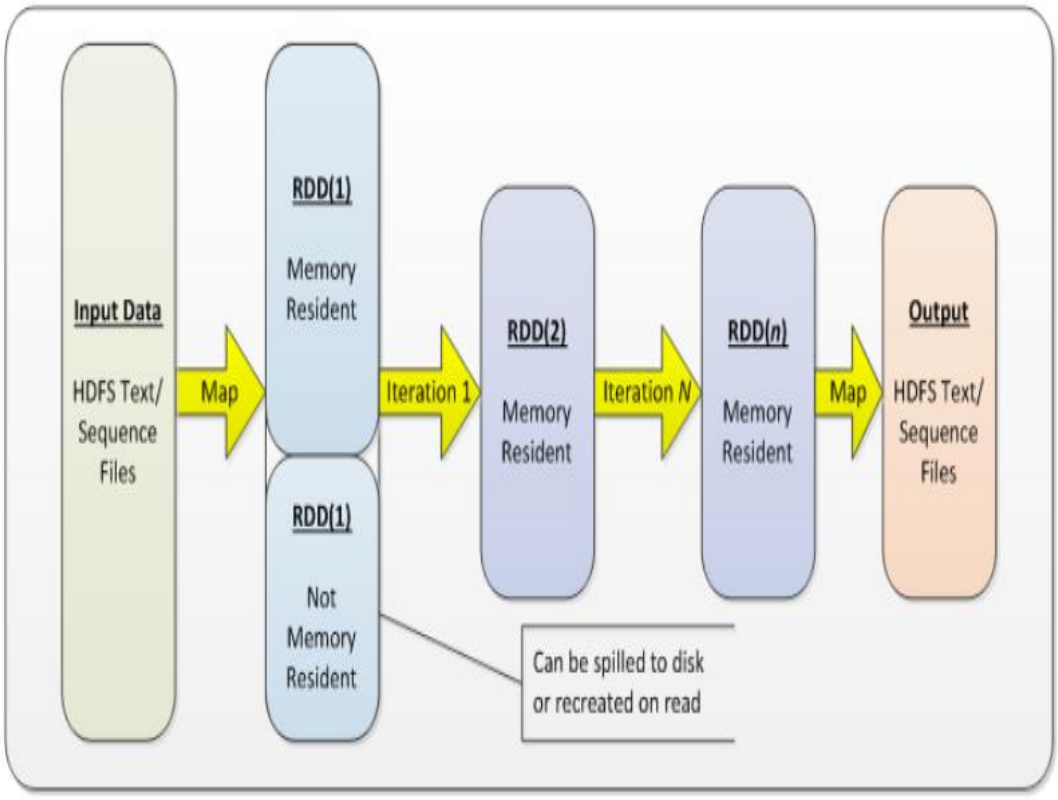

Рисунок 17 – Схема обробки даних

На відміну від Hadoop, SPARK і Scala утворюють тісну інтеграцію, при якій Scala може легко маніпулювати розподіленими наборами даних як локальними розподіленими об'єктами. SPARK призначений для вирішення ітеративних задач з розподіленими даними, він може працювати з HDFS. SPARK розроблений в лабораторії Каліфорнійського університету в Берклі для побудови великомасштабних і швидкодіючих додатків аналізу даних.

Основним поняттям в APACHE SPARK є RDD (Resilient Distributed Dataset з англ. Еластичний розподілений набір даних), схема обробки колекцій представлена на рисунку 14, який представляє собою колекцію, над якою можна робити перетворення двух типів (і, відповідно, вся робота з цими структурами полягає в повартоовності цих двох дій): трансформації - розподілені аналоги широко поширених перетворень колекцій в різних мовах програмування, такі як .Map (),

.filter (), .distinct (), і дії, які призводять до матеріалізації колекції: .save (), .Reduce(),. zip (). Не дивлячись на те, що для программіста RDD є локальною колекцією, APACHE SPARK забезпечує те, що її вміст буде рівномірно розподілено по всім машинам в обчислювальному кластері і всі операції будуть проводитися в пам'яті до тих пір, поки не буде викликано одну із матеріалізующих операцій.

Програми запускаються як незалежні набори процесів на кластері, координуються об'єктом SPARKContext у вашій основній програмі (називається програма драйвера).

Зокрема, для роботи в кластері, SPARKContext може підключатися до декількох типів менеджерів кластерів (власний окремий менеджер кластерів SPARK, Mesos або Yarn), які розподіляють ресурси між додатками. Після з'єднання SPARK запускає Worker на вузлах кластера, які є процесами, виконують обчислення та зберігають дані. Далі, він розміщує код програми (визначений файлами JAR або Python, переданими SPARKContext) виконавцям на вузлах кластера. Після запуску SPARKContext надсилає завдання виконавцям для запуску скомпільованого коду. Схема виконання програми представлена на рисунку 18.

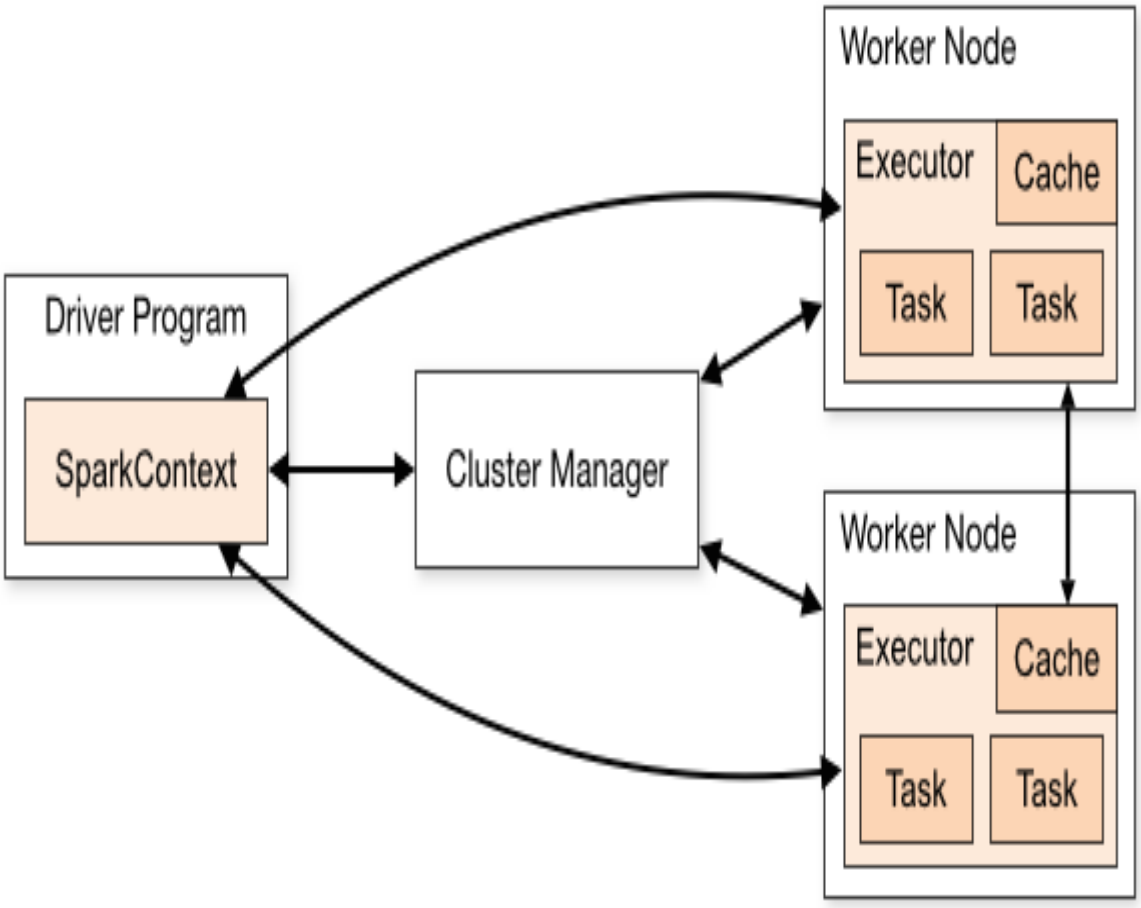

Рисунок 18 – Схема виконання програми у моделі APACHE SPARK

Кожна програма отримує власні процеси-виконавці, які зберігаються протягом усього застосування та виконують завдання в декількох потоках. Це полегшує ізоляцію програм, як на стороні планування (кожен драйвер розкладає свої власні завдання) так і на стороні виконавця (завдання різних додатків запускаються у різних JVM).

Однак це також означає, що дані не можуть бути розподілені між різними програмами SPARK (випадки SPARKContext), не записуючи їх у зовнішню систему зберігання даних.

SPARK є контейнером для основного менеджера кластерів. До якого можна додавати виконавців, які взаємодіють один з одним, досить легко запустити його

навіть в менеджері кластерів, що також підтримує інші програми (наприклад, Mesos / Yarn).

Програма драйвера повинна слухати та приймати вхідні з'єднання від виконавців протягом усього терміну. Оскільки драйвер планує завдання на кластері, його варто запускатив в одній локальній мережі із робочими вузлами. В даний час система підтримує три менеджера кластерів:

- Автономний - простий кластерний менеджер, що входить до складу SPARK, що дозволяє легко налаштувати кластер.
- APACHE Mesos - загальний менеджер кластерів, який також може запускати програми Hadoop MapReduce та сервіси.
- Hadoop Yarn - менеджер ресурсів в Hadoop 2.
- Kubernetes - це система з відкритим кодом для автоматизації розгортання, масштабування та управління контейнеризованими додатками.

Порівняння APACHE Hadoop та APACHE SPARK. SPARK - це платформа для написання додатків з відкритим кодом. Він забезпечує більш швидкий і більш загальний механізм обробки даних. SPARK в основному призначена для швидкого обчислення. Він також охоплює широкий діапазон робочих навантажень - наприклад, пакетне, інтерактивне, ітеративне та потокове.

Hadoop MapReduce: це також відкрита платформа для написання додатків. Він також обробляє структуровані та неструктуровані дані, які зберігаються в HDFS. Hadoop MapReduce спроектований таким чином, щоб обробляти великий обсяг даних на кластері. MapReduce може обробляти дані тільки в пакетному режимі.

APACHE SPARK забезпечує як пакетну обробку, так і для обробку потоку. APACHE SPARK прискорює пакетну обробку даних завдяки оптимізації обчислень та обробці в оперативній пам'яті. APACHE SPARK також може працювати з

Hadoop та його модулями. Завдяки можливості обробки даних у реальному часі SPARK є найкращим вибором для великих аналітичних даних. Еластичний розподілений набір даних (RDD) дозволяє прозоро зберігати дані в пам'яті і відправляти на диск лише те, що необхідно. У результаті зберігається багато часу, витраченого на читання та запис даних на диск.

Оскільки APACHE MapReduce зберігає дані на диску, він може обробляти великі набори даних, але відбувається втрата часу на читання та запис даних на диск. Отже, APACHE SPARK може обробляти дані в реальному часі, тобто дані, що надходять із потоків у режимі реального часу. MapReduce не призначений для обробки даних у режимі реального часу, його доцільно використовувати для виконання пакетної обробки великих обсягів даних.

1.5 Висновок

В ході роботи проаналізовано архітектуру бездротових сенсорних мереж. Безпровідна сенсорна мережа скадається з мотів, будову мотів також проаналізовано, що з’єднані між собою та серверами, де відбуваються обчислення даних сенсорів. Були описані стандарт IEEE 802.15.4 та протокол ZigBee, засобами яких моти з’єднані між собою та серверами у мережу. Схема такого зв'язку представлена на рисунку 19.

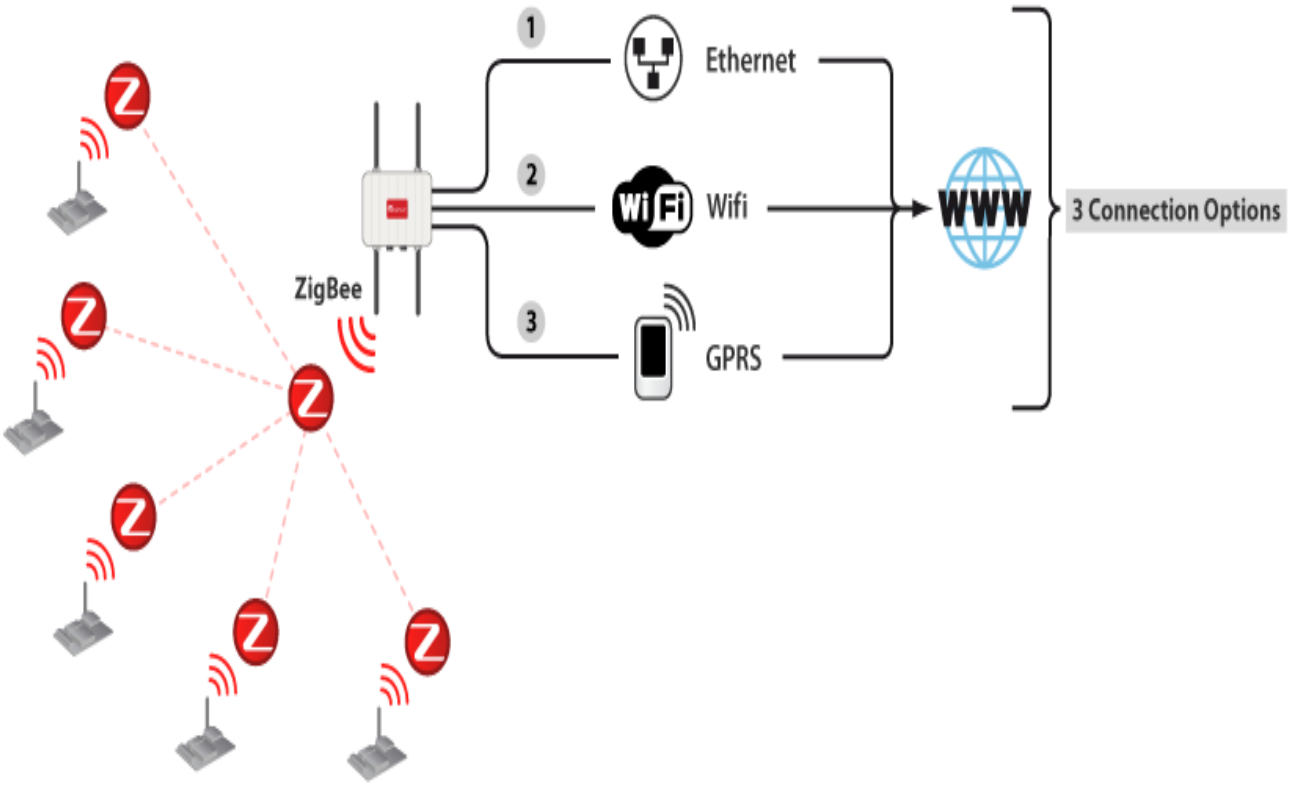

Рисунок 19 – Схема з'єднання мотів засобами протоколу ZigBee.

Також розглянуто метод TDOA (Time Difference of Arrival) для локалізації об’єктів в розподілених сенсорних мережах та алгоритм розрахунку координат джерела акустичного сигнала у мережі сенсорів. Наведені графіки визначення різниці часу приходу сигналів та ідентифікації джерела за сигналом.

Проаналізовано доцільність організації кластера серверів для обчислень даних серверів та вимоги до систем розподілених обчислень.

А також проаналізовано архітектуру сучасних платформи для обчислень великих колекцій даних. Вибрано платформу APACHE SPARK для подальшої розробки. Платформа APACHE SPARK відповідає вимогам зазначеним у пункті 1.3. Такий вибір обумовлений тим, що для задачі обчислення координат необхідно виконувати обробку даних в режимі реального часу, а платформа APACHE SPARK використовує оперативну пам’ять, а не дисковий простір, при обробці даних, що

збільшує продуктивність розподілених обчислювальних систем побудованих на цій платформі.

Також вагомою перевагою э те, що SPARK складається з програми драйвера, яка запускає основну функцію користувача та виконує різні паралельні операції в кластері. Основна абстракція SPARK забезпечує еластичний розподілений набір даних (RDD), який являє собою набір елементів, розділених між вузлами кластера, які можуть працювати паралельно.

Отже, для обчислень даних сенсорів буде використовуватись кластер серверів під управлінням фреймворку APACHE SPARK і матиме структуру зображену на рисунку 20.

Задача магістерської роботи полягає в оцінці можливості збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом в розподіленій мережі сенсорів.

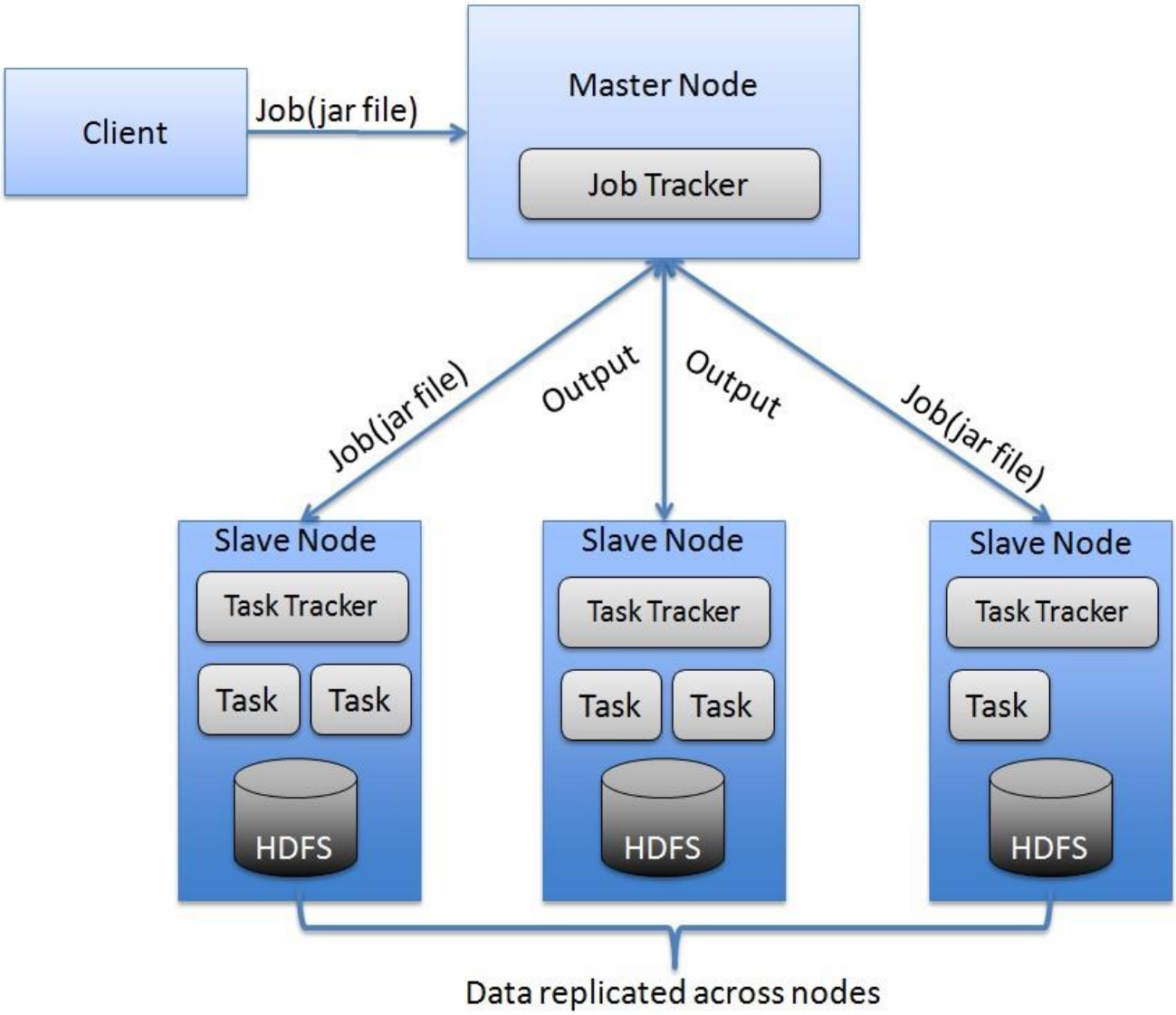

Рисунок 20 – Кластер серверів під управлінням фреймворку APACHE SPARK

2 РЕАЛІЗАЦІЯ МЕТОДУ TDOA НА ПЛАТФОРМІ APACHE SPARK

2.2 Аналіз вимог до розробленого додатку локалізації об’єктів за акустичним сигналом у мережі сенсорів.

Задача полягає в оцінці можливості збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом в розподіленій мережі сенсорів, шляхом забезпечення системи розподіленої обробки даних сенсорів. З метою вирішення поставленої задачі побудовано модель, що імітує локалізацію джерел акустичних сигналів в сенсорній мережі та складається з мережі сенсорів та серверів, що утворють кластер, а також розроблено алгоритм, що дозволить використати метод TDOA (Time Difference of Arrival) для локалізації сигналів та розподілити обчислення між серверами кластера реалізуючи модель APACHE SPARK. Отже, додаток має забезпечити наступний функціонал:

- можливість запуску на одному сервері та на декількох серверах;

- приймати безперервний потік даних сенсорів;

- здійснювати фільтрацію даних - залишати дані лише від тих сенсорів, які зафіксували сигнал від джерела;

- здійснювати розподіл даних між вузлами кластера, за ідентифікатором джерела сигналу;

- паралельно здійснювати обчислення координат джерела сигналу за згрупованими даними на кожному вузлі кластера;

- сформувати вихідний потік даних;

- зберегти результати обчислень і ідентифікатор сигналу.

Ідентифікація джерела сигналу відбувається не є предметом довартожень і повинна відбуватись на етапі формування вхідного потоку. Для спрощення моделі додаток приймає вхідні дані з ідентифікованим джерелом.

Схема роботи розробленого додатку зображена на рисунку 20.

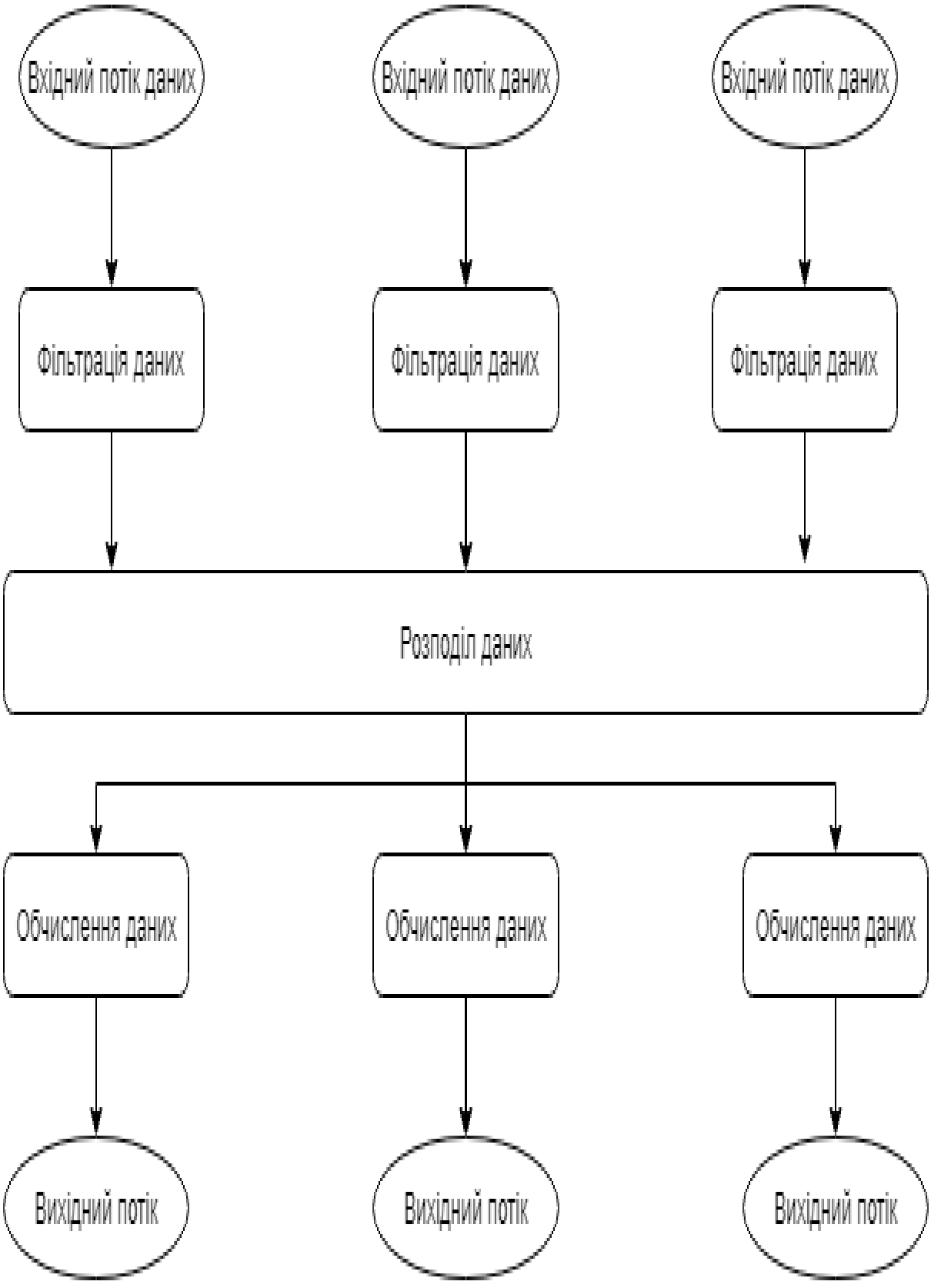

Рисунок 20 – Схема роботи розробленого додатку.

2.3Аналіз вхідного потоку даних та етапів розподілу обчислень на вузлах кластера

На першому етапі додаток на кожному вузлі приймає вхідну коллекцію даних у вигляді JSON об'єкта зображеного на рисунку 20.


```
[
{
  'x': '1',
  'y': '1',
  'z': '5',
  'timestamp': '23224232342',
  'signalId': '1',
},
{
  'x': '1',
  'y': '5',
  'z': '5',
  'timestamp': '24224232342',
  'signalId': '12',
},
{
  'x': '12',
  'y': '2',
  'z': '16',
  'timestamp': '25224232342',
  'signalId': '0',
}
{
  'x': '3',
  'y': '4',
  'z': '5',
  'timestamp': '25344232342',
  'signalId': '3',
}
{
  'x': '0',
  'y': '0',
  'z': '0',
  'timestamp': '26224232342',
  'signalId': '3',
},
{
  'x': '1',
  'y': '7',
  'z': '5',
  'timestamp': '27224232342',
  'signalId': '3',
}
]
```


Рисунок 20 – Вхідна колекція даних

Де поля ‘x’, ‘y’, ‘z’ – координати сенсорів, ‘timestamp’ – час прийняття сигналу, а ‘signalId‘– ідентифікатор сигналу. Далі із вхідної колекції утворюється RDD набір. Дані RDD набору розподіляються між вузлами кластеру і на кожному з вузлів

виконується операція фільтрації даних з умовою ‘signalId’ не дорівнює ‘0’. Схема операції зображена на рисунку 21.

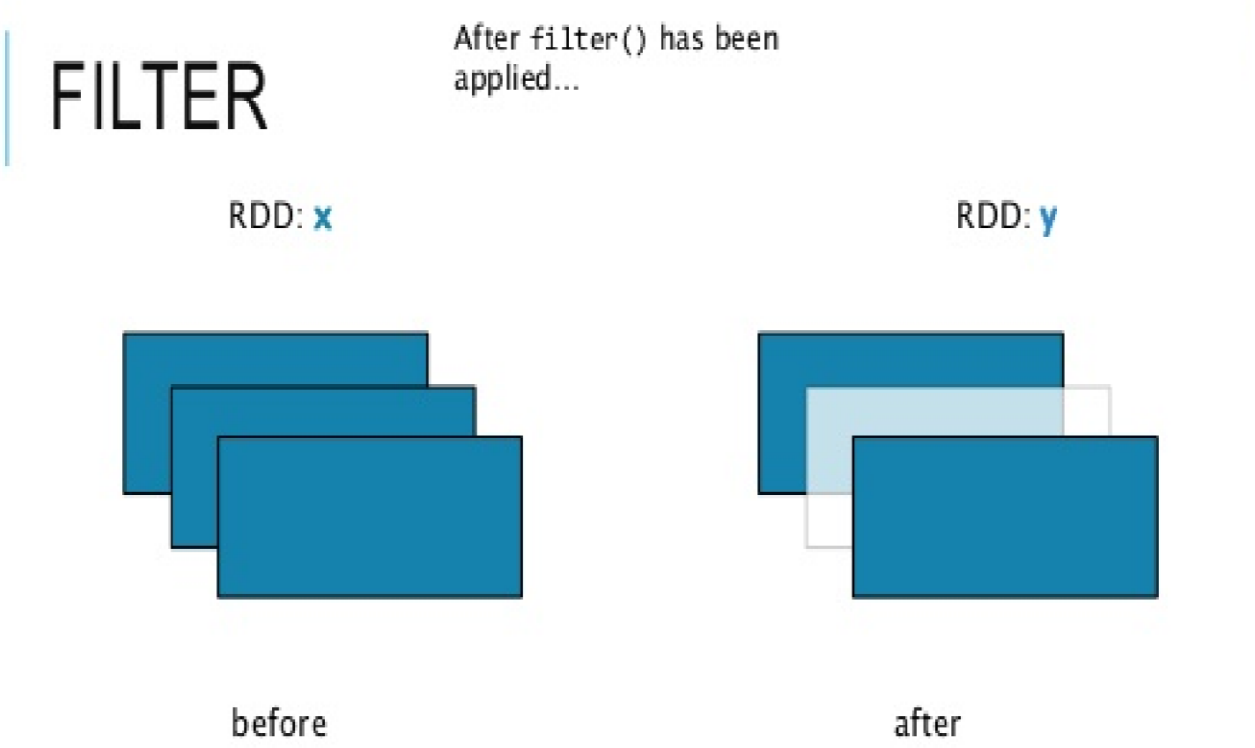

Рисунок 21 – операція RDD – filter

SPARK RDD Filter: RDD <T> клас забезпечує метод filter () для вибору тих елементів, які підпорядковуються умовам фільтра, які передаються як аргумент методу. Трансформації в SPARK здійснюються в «ледачому» режимі - тобто, результат не обчислюється відразу після трансформації. Замість цього вони просто «запам'ятовують» операцію, яку варто провести, і набір даних (напр., Файл), над яким потрібно зробити операцію. Обчислення трансформацій відбувається тільки тоді, коли викликається дію, і його результат повертається основній програмі. Завдяки такому дизайну підвищується ефективність SPARK. Наприклад, якщо великий файл був перетворений різними способами і переданий першої дії, то

SPARK обробить і поверне результат лише для першого рядка, а не стане опрацьовувати таким чином весь файл.

В результаті цієї відсіюються дані сенсорів, що не прийняли сигнал. Після даного перетворення проміжний набір має вигляд зображений на рисунку 22.


```
[
{
  'x': '1',
  'y': '1',
  'z': '5',
  'timestamp': '23224232342',
  'signalId': '1',
},
{
  'x': '1',
  'y': '5',
  'z': '5',
  'timestamp': '24224232342',
  'signalId': '12',
},
{
  'x': '3',
  'y': '4',
  'z': '5',
  'timestamp': '25344232342',
  'signalId': '3',
}
{
  'x': '0',
  'y': '0',
  'z': '0',
  'timestamp': '26224232342',
  'signalId': '3',
},
{
  'x': '1',
  'y': '7',
  'z': '5',
  'timestamp': '27224232342',
  'signalId': '3',
}
]
```


Рисунок 22 – Проміжна колекція після фільтрації

Наступною операцією виконаною над RDD набором є операція розподілу між кластерами по ідентифікатору сигналу (groupById). Цей підхід до архітектури

намагається збалансувати затримку, пропускну спроможність і відмостійкість, застосовуючи пакетну обробку для надання всебічних та точних розрізів (view) пакетів даних, одночасно використовуючи потокову обробку в режимі реального часу для створення розрізів операційних даних. Ці два розрізи даних можуть бути з'єднані перед виводом. Принцип дії такої операції зображено на рисунку 23.

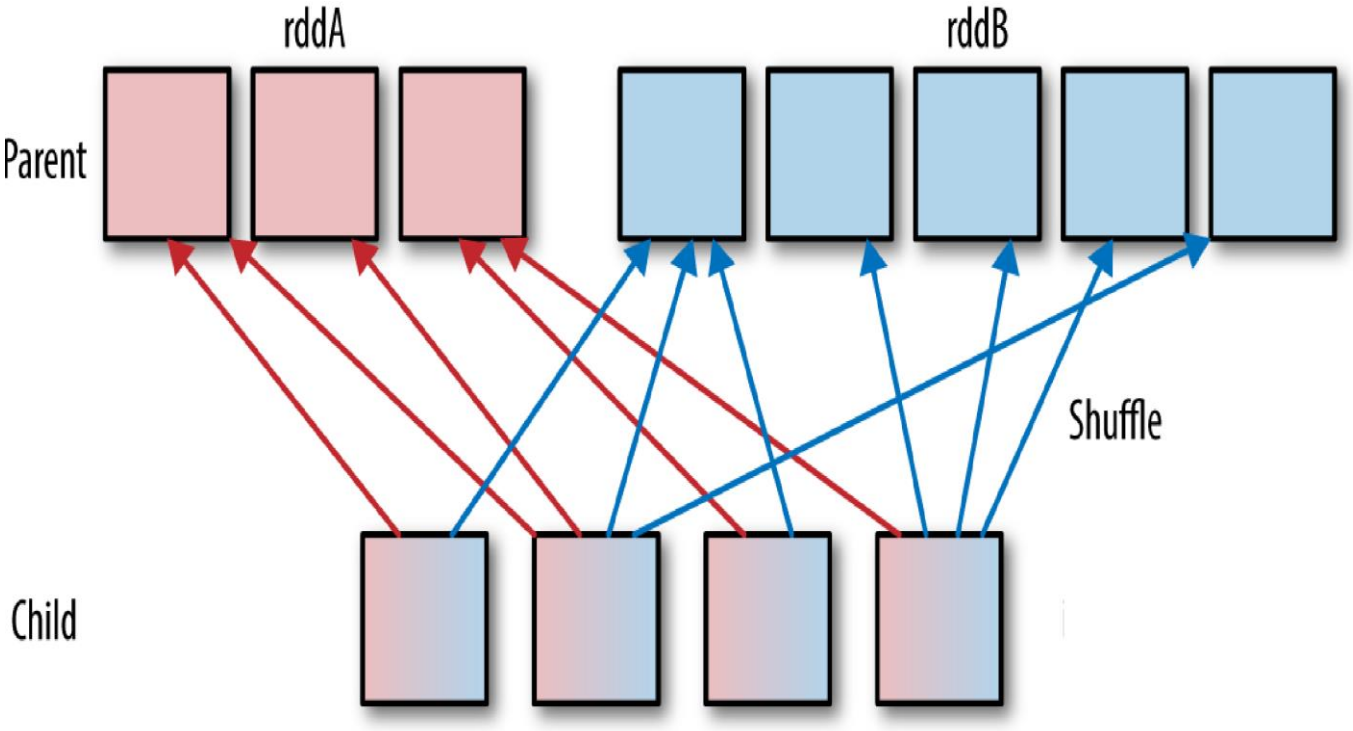

Рисунок 23 – операція RDD – groupByKey

На етапі розподілу даних відбувається сортування відфільтрованих даних з усіх вузлів по ключу та обмін даними між вузлами таким чином, щоб на кожному вузлі опинились дані з однаковими ключами. Після виконання операцій на кроці розподілу даних ми маємо на кожному вузлі всі необхідні дані для обчислення координат тих джерел сигналів, дані з ключами яких були збережені на поточному вузлі. Ефективним є використання асинхронного I/O, тобто такого, який не блокує інші операції (тобто потік виконання) під час очікування завершення передачі.

Наприклад, на Java Virtual Machine можна використовувати можливості Java NIO (New IO), які реалізують асинхронний ввід/вивід за допомогою опитування (polling) джерел даних. При застосуванні цього методу, якщо очікуються дані із мережі, періодично перевіряється стан мережевого сокету на предмет їх надходження; при цьому потік не блокується, інші обчислення та запити продовжують виконуватися. В результаті перетворень отримано новий набір даних зображених на рисунку 24.


```
[
  'signalId': '1'
  {
    'x': '1',
    'y': '1',
    'z': '5',
    'timestamp': '23224232342',
    'signalId': '1',
  },
],

[
  'signalId': '12',
  {
    'x': '1',
    'y': '5',
    'z': '5',
    'timestamp': '24224232342',
    'signalId': '12',
  },
],


[
  'signalId': '3',
  {
    'x': '3',
    'y': '4',
    'z': '5',
    'timestamp': '25344232342',
    'signalId': '3',
  }
  {
    'x': '0',
    'y': '0',
    'z': '0',
    'timestamp': '26224232342',
    'signalId': '3',
  },
  {
    'x': '1',
    'y': '7',
    'z': '5',
    'timestamp': '27224232342',
    'signalId': '3',
  }
]
```


Рисунок 24 – Проміжні набори згруповані на вузлах кластера за ідентифікатором сигналу

Після розподілу даних за ключем відбувається виконання «згортки» (Map) даних – на кожному із вузлів паралельно відбувається обчислення координат джерел сигналів згідно методу TDOA для кожного згрупованого за ключем набору даних. Стадія Map, на якій дані обробляються за допомогою функції Map(), яку визначає користувач. Робота на цій стадії полягає у переробці та фільтрації даних. Робота операції схожа на метод Map() у функціональних мовах програмування – функція застосовується до кожного елементу списку. Функція Map приймає список на вході і повертає множину пар ключ-значення. Схема оперції «згортки» зображена на рисунку 25.

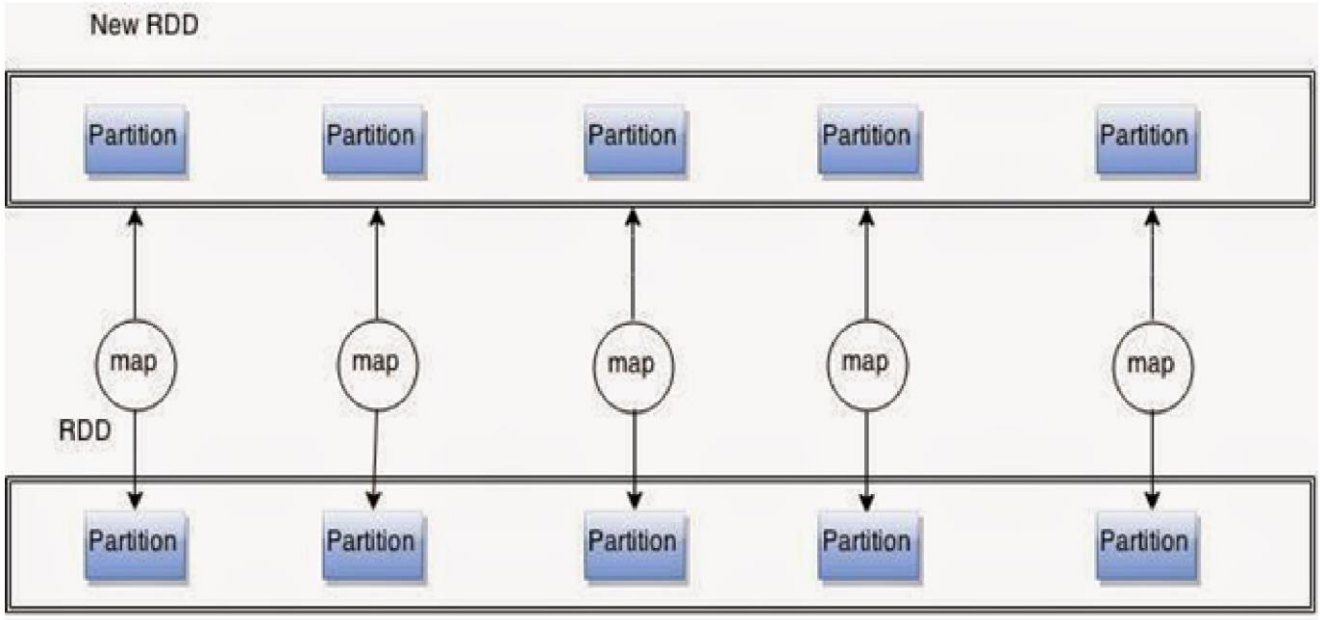

Рисунок 25 – операція RDD – Map

Після виконання операції «згортки» ми отримаємо новий набір даних зображений на рисунку 26.

```
[
  {
    'x': '1',
    'y': '1',
    'z': '5',
    'signalId': '1',
  }
],

[
  {
    'x': '1',
    'y': '5',
    'z': '5',
    'signalId': '12',
  }
],


[
  {
    'x': '3',
    'y': '4',
    'z': '5',
    'signalId': '3',
  }
]
```


Рисунок 25 – Кінцеві набори готові до збереження

Де поля ‘x’, ‘y’, ‘z’ – координати сенсорів, ‘timestamp’ – час прийняття сигналу, а ‘signalId‘– ідентифікатор сигналу. Далі буде визвана операція RDD collect і результуючі набори даних будуть збережені на одному з вузлів кластера. Коли операція збирання видається на RDD, набір даних копіюється у драйвер, тобто основний вузол. Враховуючи фізичний план виконання, драйвер повинен координувати планування індивідуальних задач виконавцям виконавці починають працювати, вони реєструються драйверу, тому він має інформацію про дії виконавців протягом всього проміжку часу. Кожен виконавець являє собою процес, здатний виконувати завдання і зберігати дані RDD. Драйвер відстежує поточний набір виконавців і розпланувує кожне завдання у відповідному місці на основі розміщення даних. Драйвер також відстежує місце зберігання керованих даних і використовує цю інформацію для планування майбутніх задач, які звертаються до цих даних. Принцип дії команди collect зображено на рисунку 26.

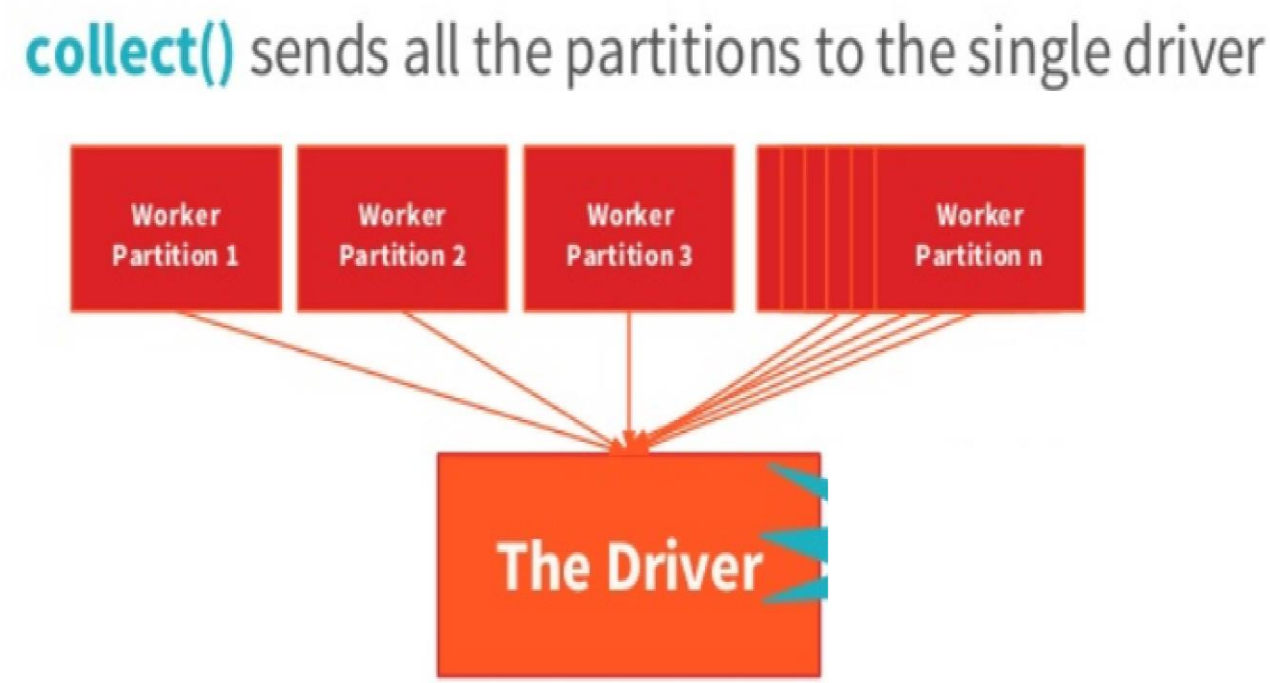

Рисунок 25 – операція RDD – collect

2.3 UML діаграма класів програми

Діаграми варіантів використання показують взаємодії між варіантами використання і діючими особами, відображаючи функціональні вимоги до системи з точки зору користувача. Мета побудови - документування функціональних вимог в загалом вигляді (вимога - простота). Варіант використання - повартоовність дій (транзакцій), виконуваних системою у відповідь на подію, що ініціюється деяким зовнішнім об'єктом (дійовою особою). На рисунку 26 зображено UML діаграму класів програми.

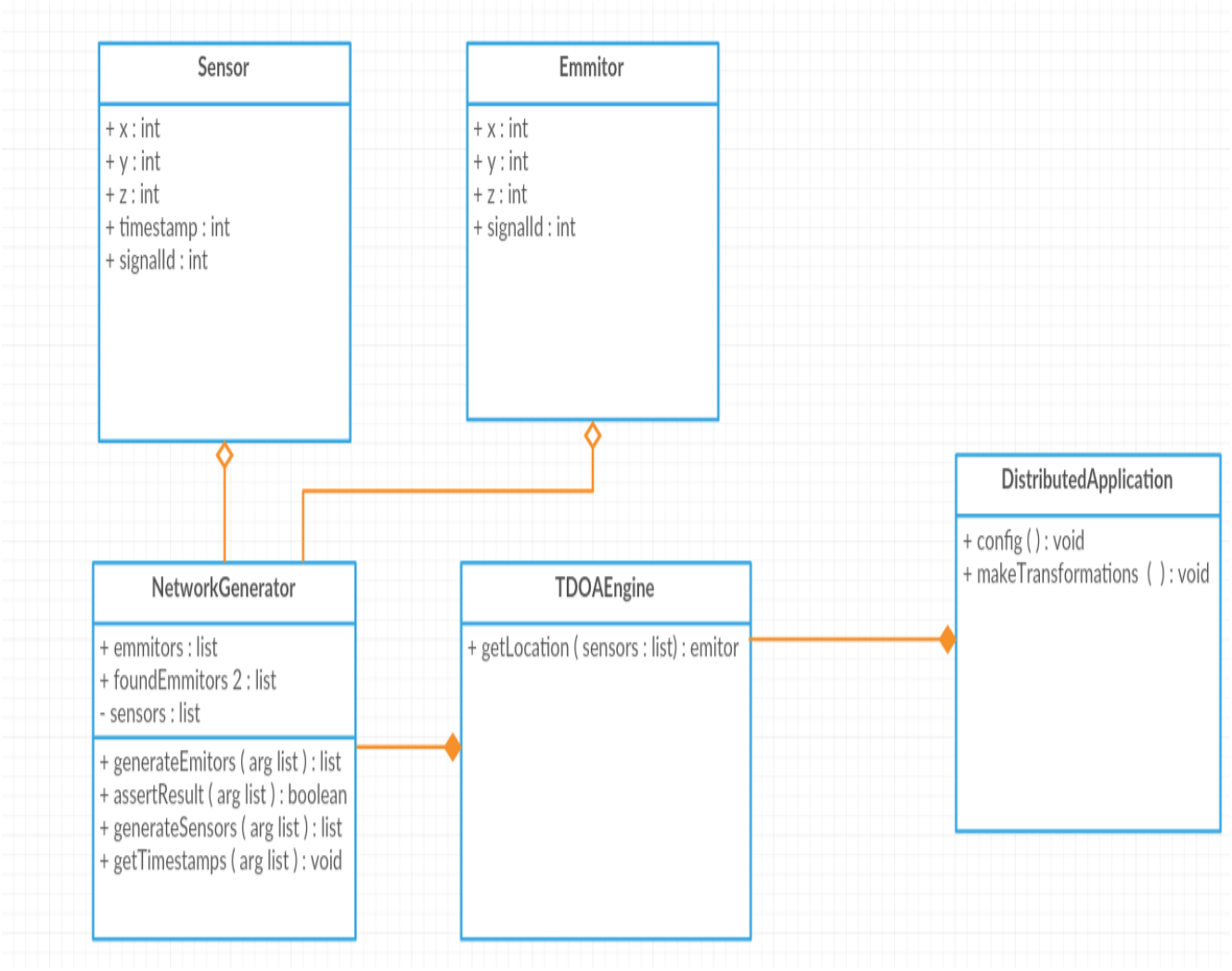

Рисунок 26 – UML діаграма класів

Класс NetworkGenerator – генерує вхідний потік даних та ініціалізує координати джерел та сенсорів. Класс TDOAEngine – забезпечує локалізацію об’єкта методом TDOA, на вхід отримує типізовану коллекцію координат сенсорів і час надходження сигналу, повертає координати джерела сигналу. Класс DistributedApplication є точкою входу у програму, тут встановлюються конфігурації запуску програми та описані трансформації набору даних на кластері серверів.

2.4 Висновки

У даному розділі проведено сформовано вимоги щодо функціоналу програмного забезпечення, що імітує локалізацію джерел акустичних сигналів в сенсорній мережі та складається з мережі сенсорів та серверів, що утворюють кластер, а також використовує метод TDOA (Time Difference of Arrival) для локалізації сигналів та розподілює обчислення між серверами кластера реалізуючи модель APACHE SPARK.

Сформовано декілька етапів розподілу й обробки даних та проаналізовано трансформації вхідного потоку даних на вузлах кластера на кожному із етапів розподілу й обробки.

Створено UML діаграму класів та проведено опис призначення класів.

3 ТЕСТУВАННЯ СТВОРЕНОЇ СИСТЕМИ

3.1 Компіляція та запуск програмного забезпечення

Для запуску програм на платформі APACHE SPARK необхідно переконатися, що на кожному з вузлів встановлений JVM. Java Virtual Machine (скорочено Java VM, JVM) - віртуальна машина Java - основна частина виконавчої системи Java, так званої Java Runtime Environment (JRE). Віртуальна машина Java виконує байт-код Java, попередньо створений з вихідного тексту Java-програми компілятором Java. За замовчуванням SPARK ініціалізує для JVM пам'ять об'ємом 512 МБ. Щоб уникнути помилки OOM, SPARK дозволяє використовувати тільки 90% купи, яка контролюється параметром SPARK.storage.safetyFraction SPARK. SPARK дозволяє зберігати деякі дані в пам'яті, яку використовує пам'ять для свого кеша LRU. Тому для кешування даних, які обробляються, зарезервований певний обсяг пам'яті, і ця частина зазвичай становить 60% від безпечної купи, яка контролюється параметром SPARK.storage.memoryFraction. Перевірка на встанолення JVM показана на рисунку 27 та рисунку 28.

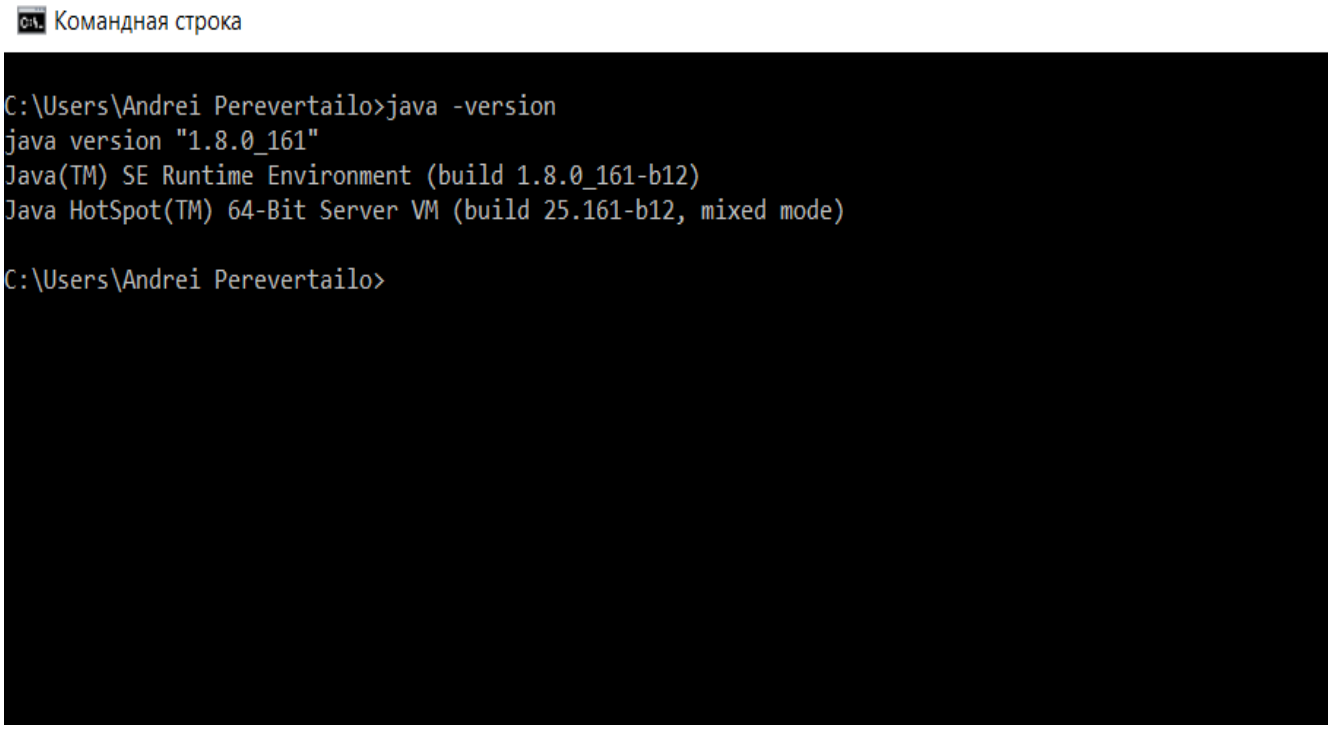

Рисунок 27 – Перевірка JVM в операційній системі Windows

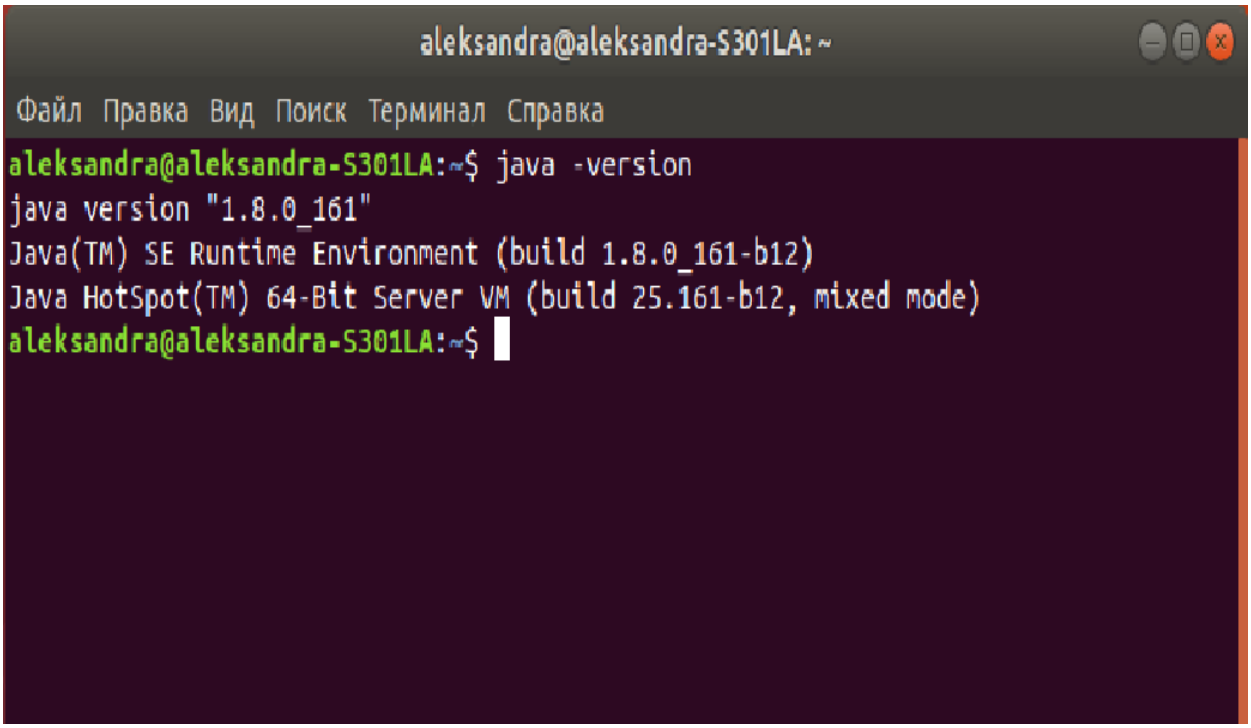

Рисунок 28 – Перевірка JVM в операційній системі Ubuntu

Якщо хоча б в одному з вузлів кластера не встановлених JVM, його необхідно встановити. Встановлення JVM на операційній системі продемонстровано на рисунку 29, встановлення JVM на операційній системі Windows описано на офіційному сайті компанії Oracle.

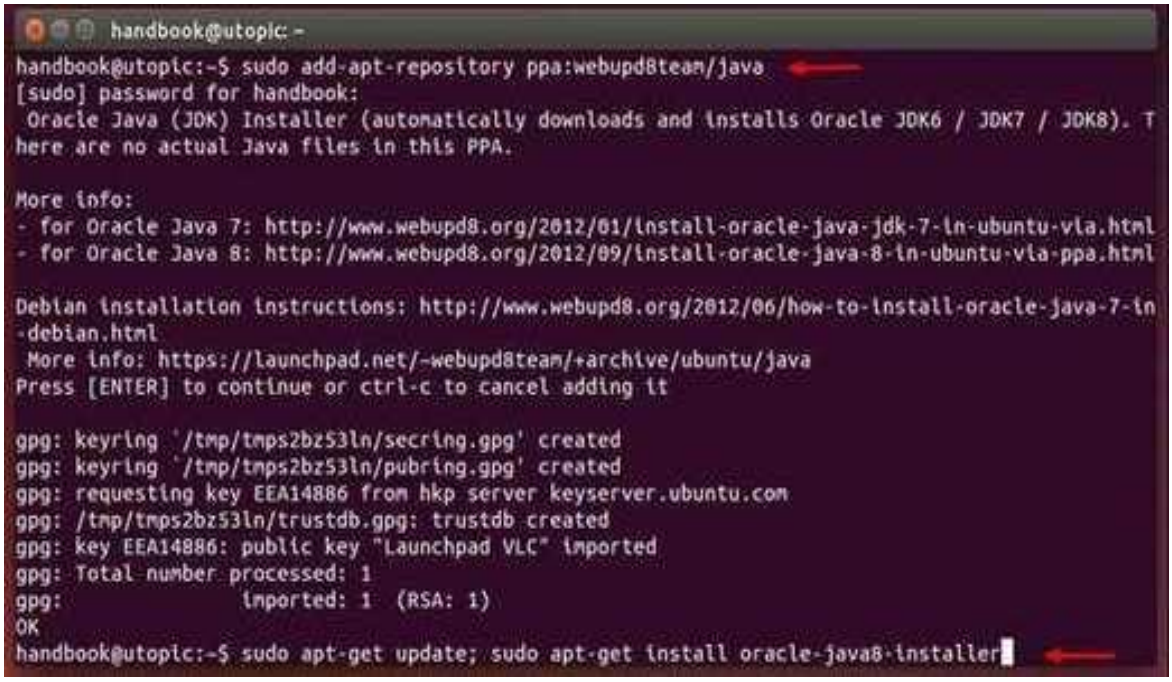

Рисунок 29 – Встановлення JVM в операційній системі Ubuntu

Для компіляції розробленого додатку необхідно встановити APACHE Maven. APACHE Maven - фреймворк для автоматизації компіляції проектів на основі опису їх структури в файлах на мові POM, що є підмножиною XML. Перевірка чи встановлений APACHE Maven зображена на рисунку 30 та рисунку 31.

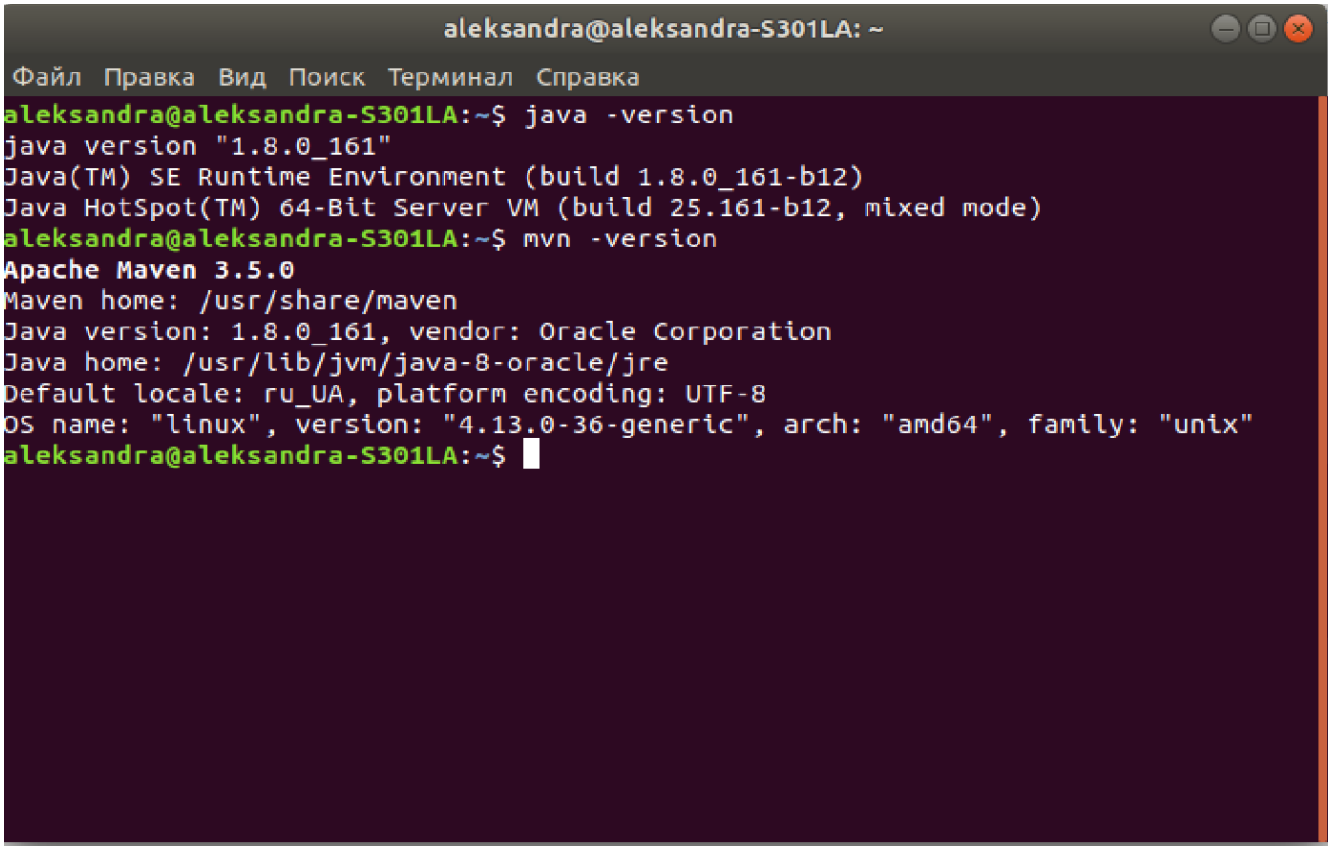

Рисунок 30 – Перевірка APACHE Maven в операційній системі Ubuntu

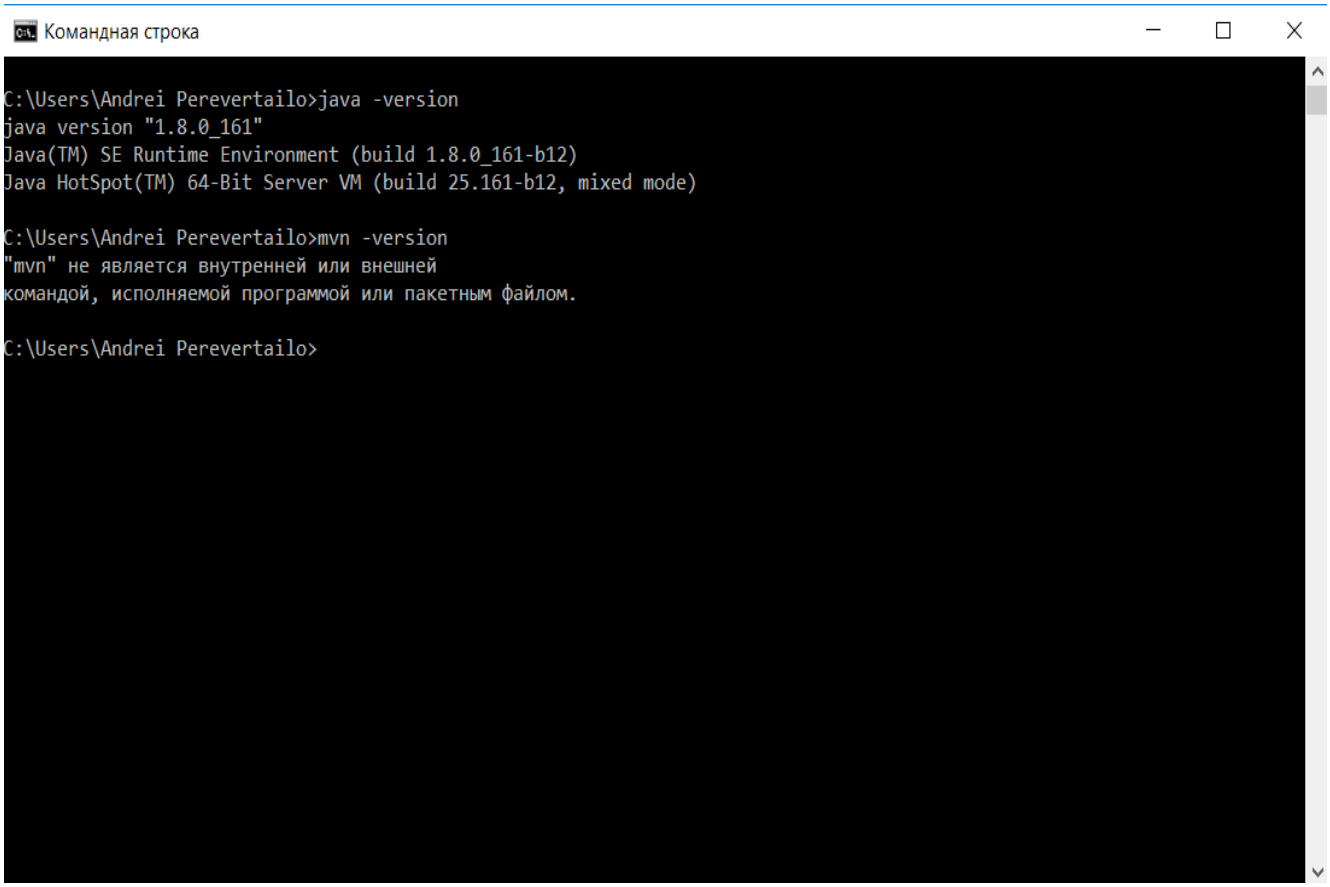

Рисунок 31 – Перевірка APACHE Maven в операційній системі Windows

Хоча б в одному з вузлів кластера має буди встановлений APACHE Maven для компіляції вихідного коду, а також для тестування коду та завнтаження додаткових бібліотек. Встановлення APACHE Maven на операційній системі продемонстровано на рисунку 32, встановлення JVM на операційній системі Windows описано на офіційному сайті компанії APACHE. Останню версію завжди можна завантажити на сторінці завантаження на офіційному сайті. Просто розпаковуємо архів в будь-яку директорію. Далі необхідно створити змінну в Path, в якій необхідно вказати шлях до Maven . Заходимо в Win + Pause - Додатково - Змінні середовища - у верхньому віконці натискаємо "Створити", вводимо ім'я M2_HOME та шлях до Maven-а, наприклад " C: \ APACHE - Maven - 2.2.1 ". Далі там же створюємо ще одну змінну M2 зі значенням % M2_HOME % \ bin. Так само переконуємося, що є змінна JAVA_HOME з шляхом до JDK. Її значення має бути

приблизно таким "C: \ Program Files \ Java \ jdk1.6.0_45 \ ". І нарешті в тому ж віконці створюємо /модифікуємо змінну Path, в неї необхідно просто написати % M2 %, щоб наша папка з виконуваним файлом Maven була видна з командного рядка.

Тепер необхідно перевірити працездатність нашої установки. Для цього заходимо в командний рядок і вводимо команду "mvn - version". Повинна з'явитися інформація про версії Maven.

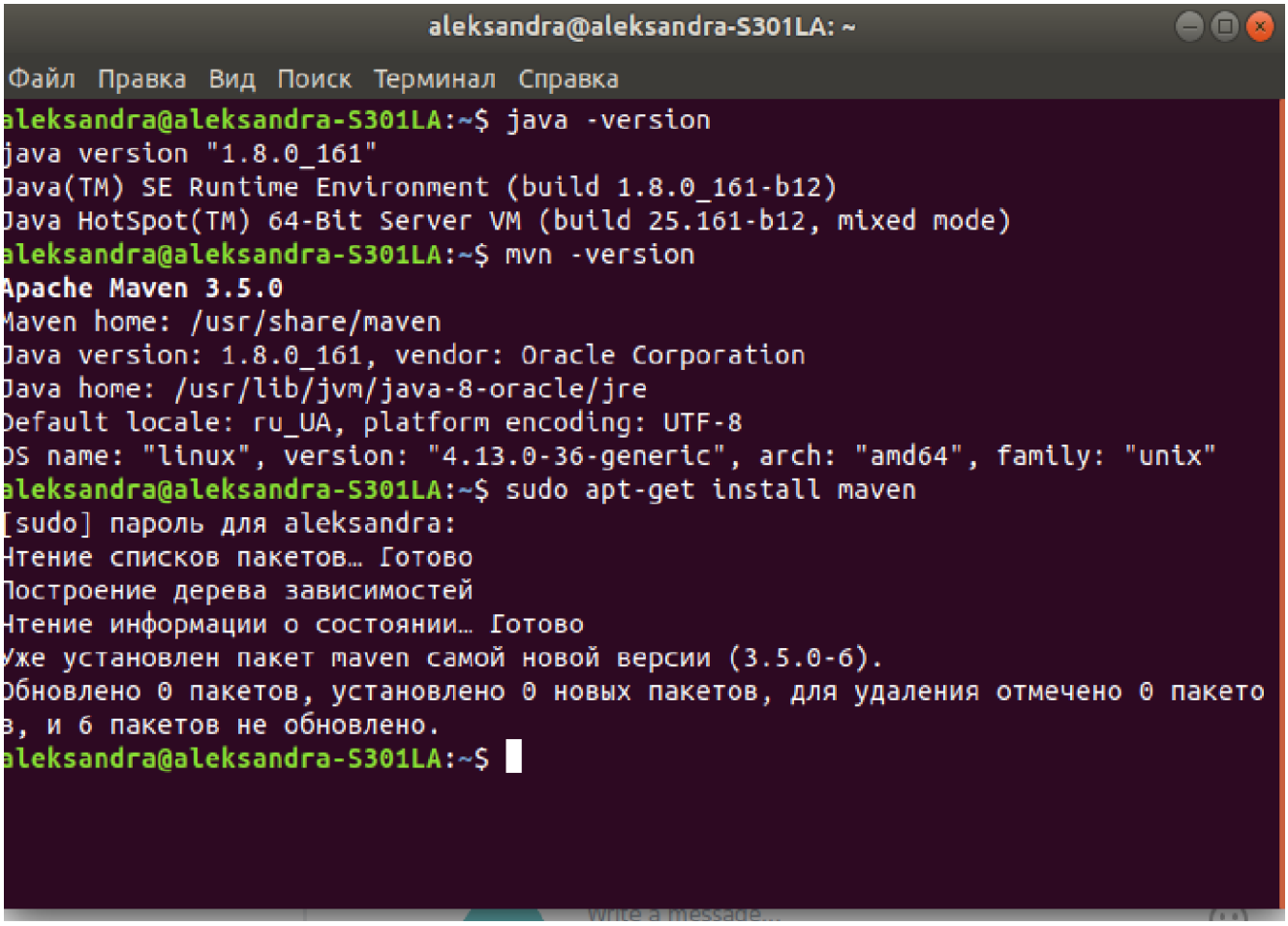

Рисунок 32 – Встановлення APACHE Maven в операційній системі Ubuntu

APACHE Maven - це фреймфорк, створений для полегшення збірки Java-проектів. Maven дозволяє прописувати усі залежності проекту в окремому файлі - "pom.xml" (Project Object Model), тобто замість завантаження усіх необхідних бібліотек у каталог проекту, достатньо дописати їх у файл. Всі описані в "pom.xml"

сторонні бібліотеки завантажуються під час збірки з Maven-репозиторію. Окрім опису залежностей, файл "pom.xml" також містить інформацію про проект та деталі конфігурації.

Переваги Maven:

- Полегшує зборку
- Створює універсальну систему збірки
- Надає інформацію про характеристики проекту
- Дозволяє легко додавати новий функціонал

Після завантаження необхідних ресурсів потрібно скомпілювати програму. Одне з основних відмінностей Maven в тому що цілі життєвого циклу компілії вже визначені, ви тільки можете викликати певну опцію або ж налаштовувати її під свої потреби. Ось список цілей:

- Validate - перевіряє коректність метаінформації про проект
- Compile - компілює вихідні коди
- Test - виконує тести класів з попереднього кроку
- Package - упаковує скомпілірование класи в зручно переміщуваний формат (jar або war, наприклад)
- Integration-test - відправляє упаковані класи в середу інтеграційного тестування і виконує тести
- Verify - перевіряє коректність пакета і задоволення вимог якості
- Install - переносить пакет в локальний репозиторій, звідки він (пекат) буде доступний для використання як залежність в інших проектах
- Deploy - відправляє пакет на віддалений production сервер, звідки інші розробники його можуть отримати і використовувати

При цьому всі кроки повартоовні. І якщо, наприклад, виконати «mvn package», то фактично будуть виконані кроки: validate, Compile, test і package.

Таким чином використовувати Maven досить просто. Написали код, виконали mvn test і можна працювати далі, переконавшись що код не містить синтаксичних і логічних помилок. Процес компіляції зображено на рисунку 33.

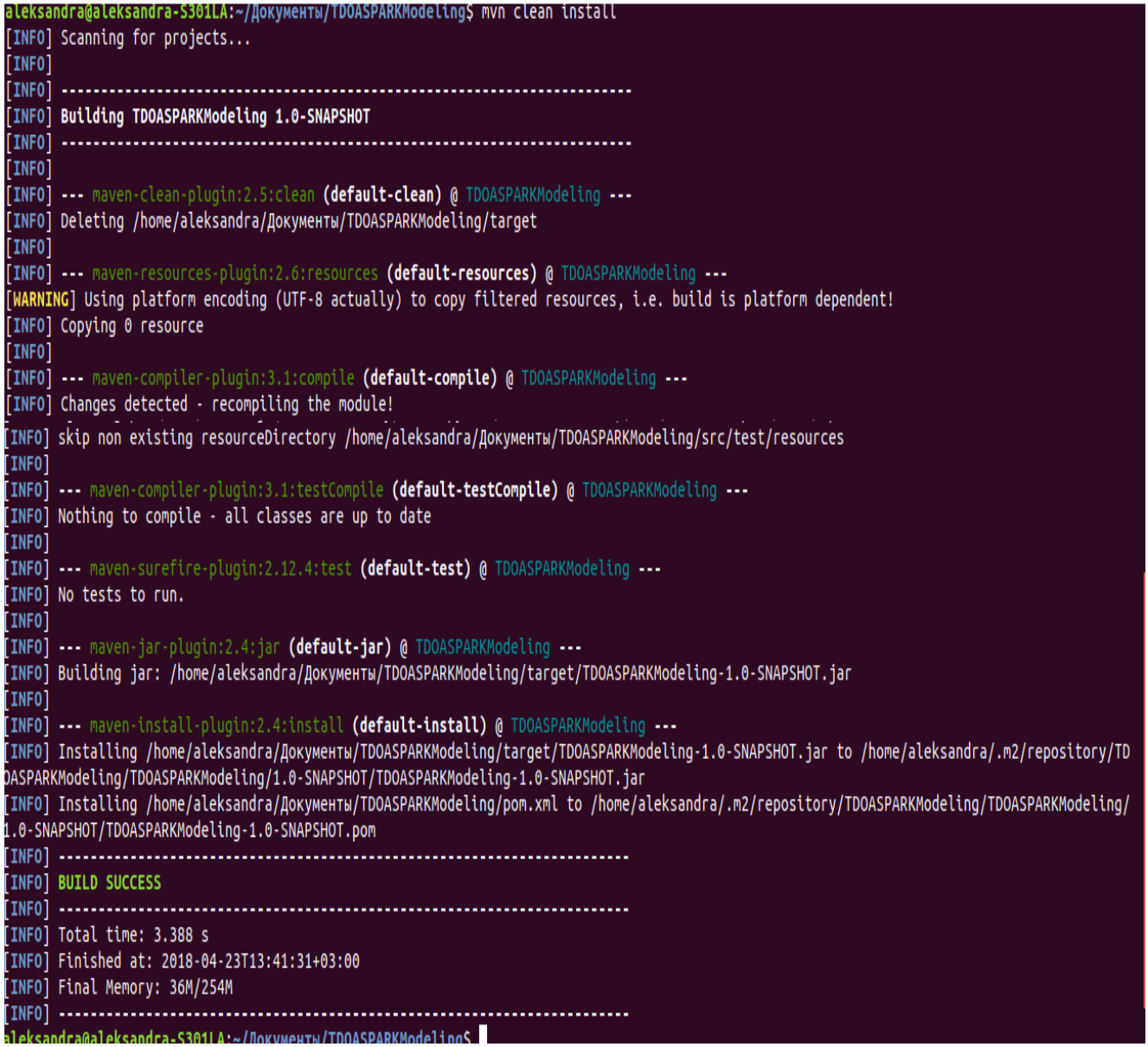

Рисунок 33 – Компіляція програми засобами APACHE Maven

Результатом виконання компіляції є утворений виконувальний файл з росширенням jar. Після виконання компіляції необхідно створити кластер серверів для виконання розподілених обчислень. Для проведення тестування програмного

забезпечення розподіленої обробки даних запуск будемо проводити на кластері з двох комп'ютерів. Для утворення кластера необхідно запустити керуючий і підлеглий сервери запуск засобами APACHE SPARK. Запуск керуючого сервера продемонстровано на рисунку 34 та рисунку 35.

Результати запуску продемонстровано на рисунках 36. SPARK драйвер відповідає за перетворення програми на виконувані модулі, що називаються задачами. Загалом всі SPARK програми навартоують одну структуру – створюють щільний розподілений набір даних з певного набору вхідної інформації, отримують нові набори даних за допомогою перетворень, а також виконують певні дії над цими наборами даних. Програма SPARK неявно створює логічний ациклічний граф операцій DAG. Коли драйвер запущений, він перетворює цей логічний граф на фізичний план виконання. SPARK виконує декілька оптимізацій, таких як «конвеєрне» об'єднання трансформацій, а також виконує перетворення графу виконання на набір етапів. Кожен етап, своєю чергою, складається з декількох задач. Завдання готуються для відправлення на кластер. Завдання є найменшою одиницею виконання у SPARK, типова програма може запустити сотні або тисячі задач. – Планування і розподіл задач виконавцям (executors).

Враховуючи фізичний план виконання, драйвер повинен координувати планування індивідуальних задач виконавцям, виконавці починають працювати, вони реєструються драйверу, тому він має інформацію про дії виконавців протягом всього проміжку часу. Кожен виконавець являє собою процес, здатний виконувати завдання і зберігати дані RDD. Драйвер відстежує поточний набір виконавців і розпланувує кожне завдання у відповідному місці на основі розміщення даних. Драйвер також відстежує місце зберігання керованих даних і використовує цю інформацію для планування майбутніх задач, які звертаються до цих даних. Драйвер надає інформацію про додаток SPARK, який працює за допомогою веб-

інтерфейсу. Виконавці (SPARK Executors) є робочими процесами, що відповідають за функціонування окремих задач у SPARK Job.

Виконавці запускаються один раз під час запуску додатка, і зазвичай працюють протягом всього життєвого циклу додатка. Виконавці виконують дві основні функції – по-перше, вони виконують завдання, які їм призначає драйвер, і повертають йому результат, а по- друге, забезпечують зберігання у пам’яті наборів даних, що кешуються користувацькими програмами, за допомогою служби Block Manager, яку містить кожен виконавець.

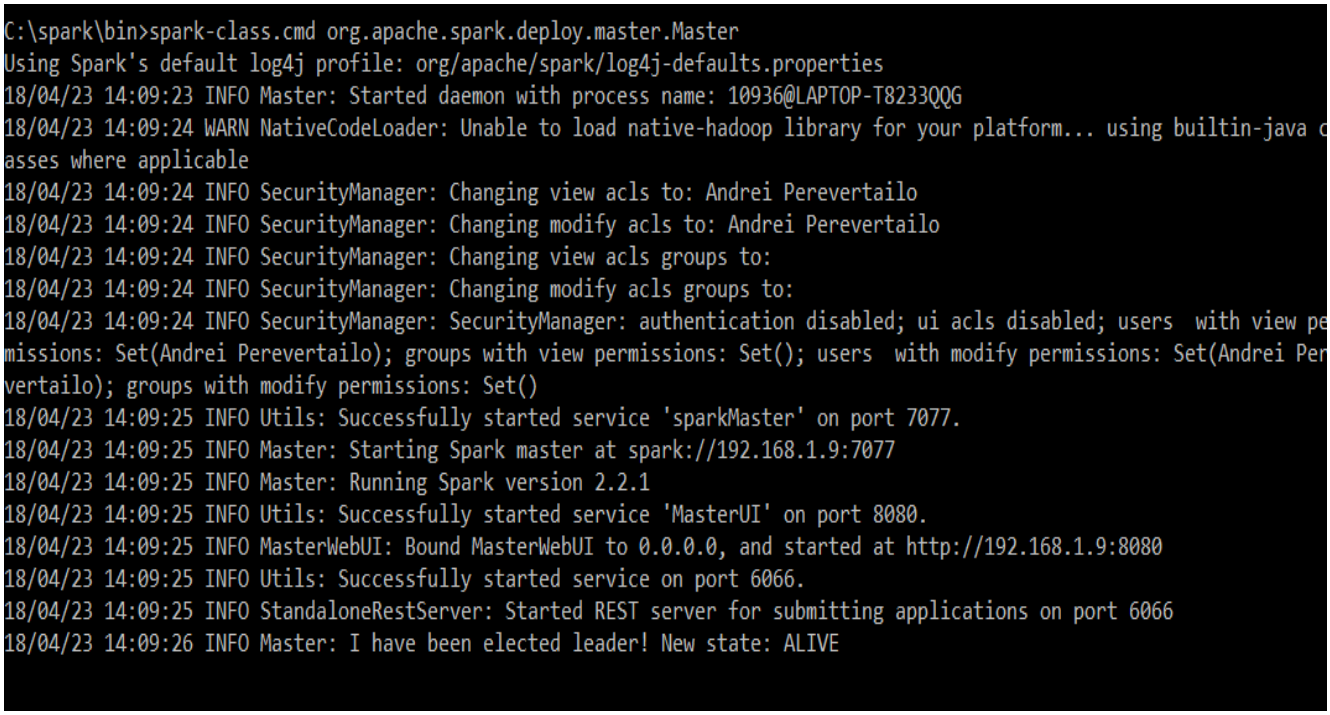

Рисунок 34 – Запуск керуючого сервера засобами APACHE SPARK в операційній системі Windows

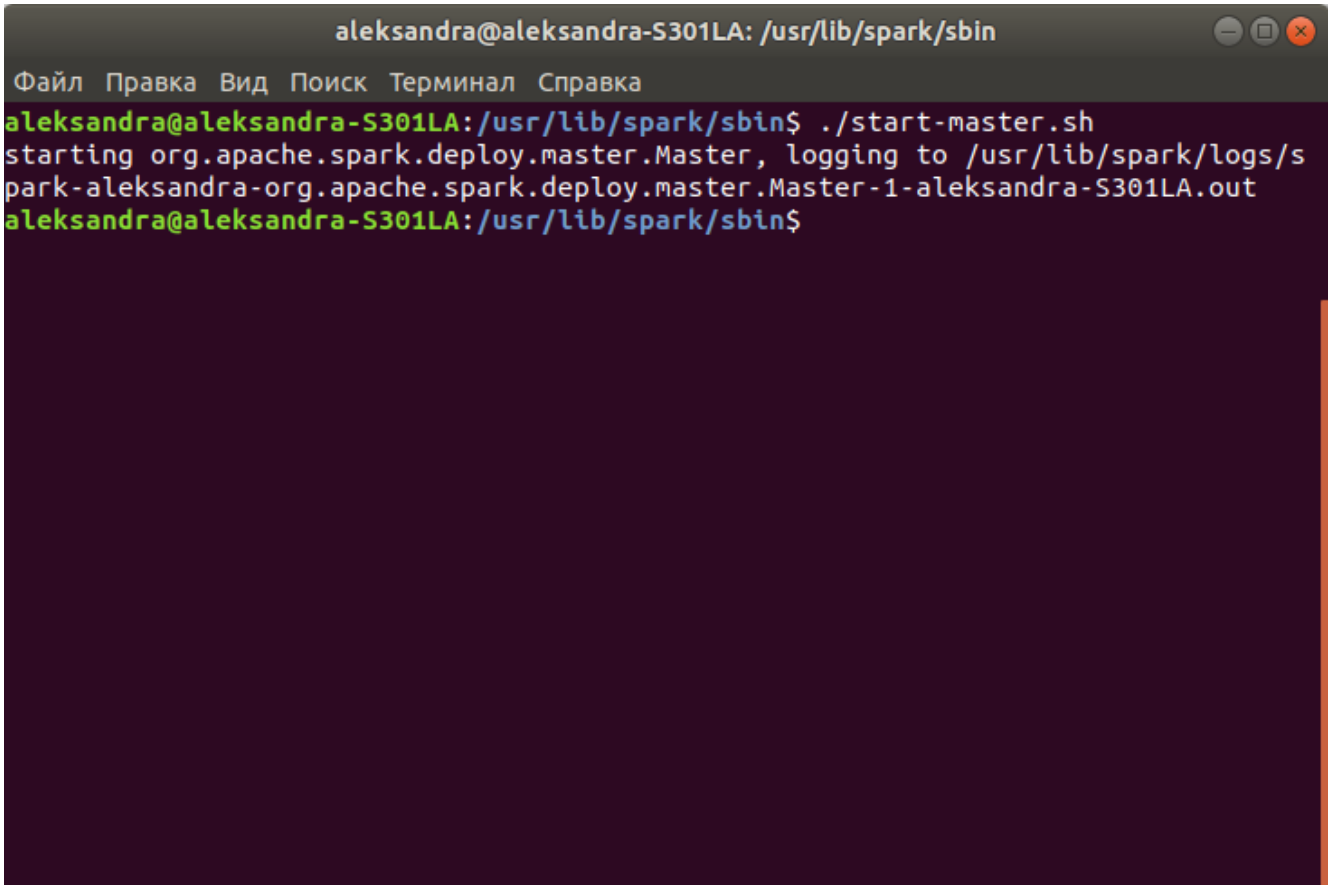

Рисунок 35 – Запуск керуючого сервера засобами APACHE SPARK в операційній системі Ubuntu

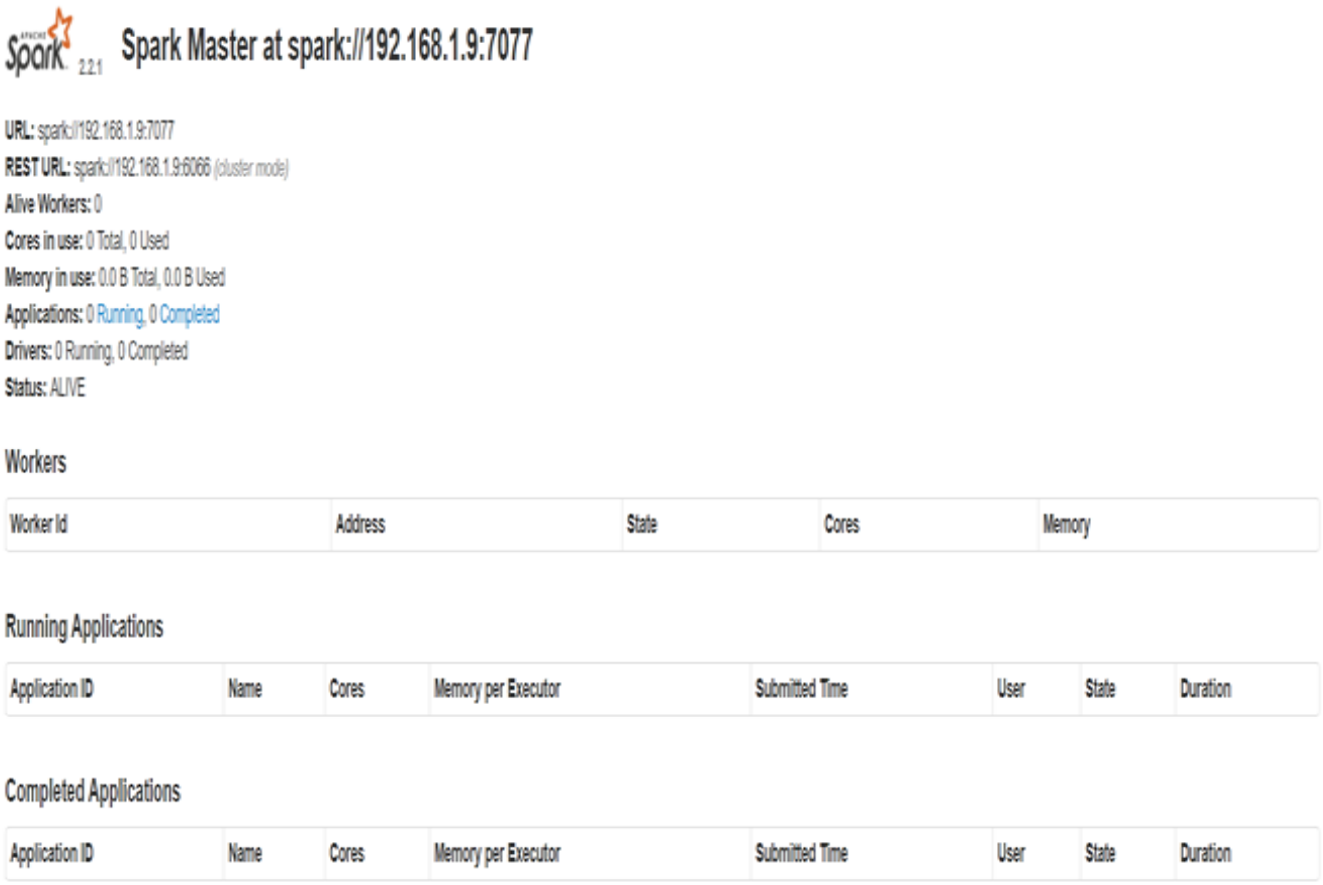

Рисунок 36 – Кластер, що складається з одного керуючого сервера.

Для виконання розподілених обчислень необхідно додати, ще два підлеглі вузли. Для роботи SPARK надає програмне API для неперервної роботи з інформаційними процесами. Модуль розподілює код по вузлах кластера, при цьому розбиває їх на задачі, створює план виконання та слідкує за його застосуванням. Результатом роботи метода SPARK є можливість запускати код через різні розподілені інформаційні системи. В режимі Standalone mode, метод управляє усіма ресурсами кластера. Yarn дозволяє додаткам запускатись на Hadoop кластері. Mesos та Local mode працюють в локальному режимі та являються альтернативними системами для управління інформаційними процесами кластера.

Так як кешування даних виконується безпосередньо всередині виконавців, завдання можуть виконуватись на кешованих розподілених наборах даних.

Процес додавання підлеглих вузлів зображено на рисунку 37 та рисунку 38.

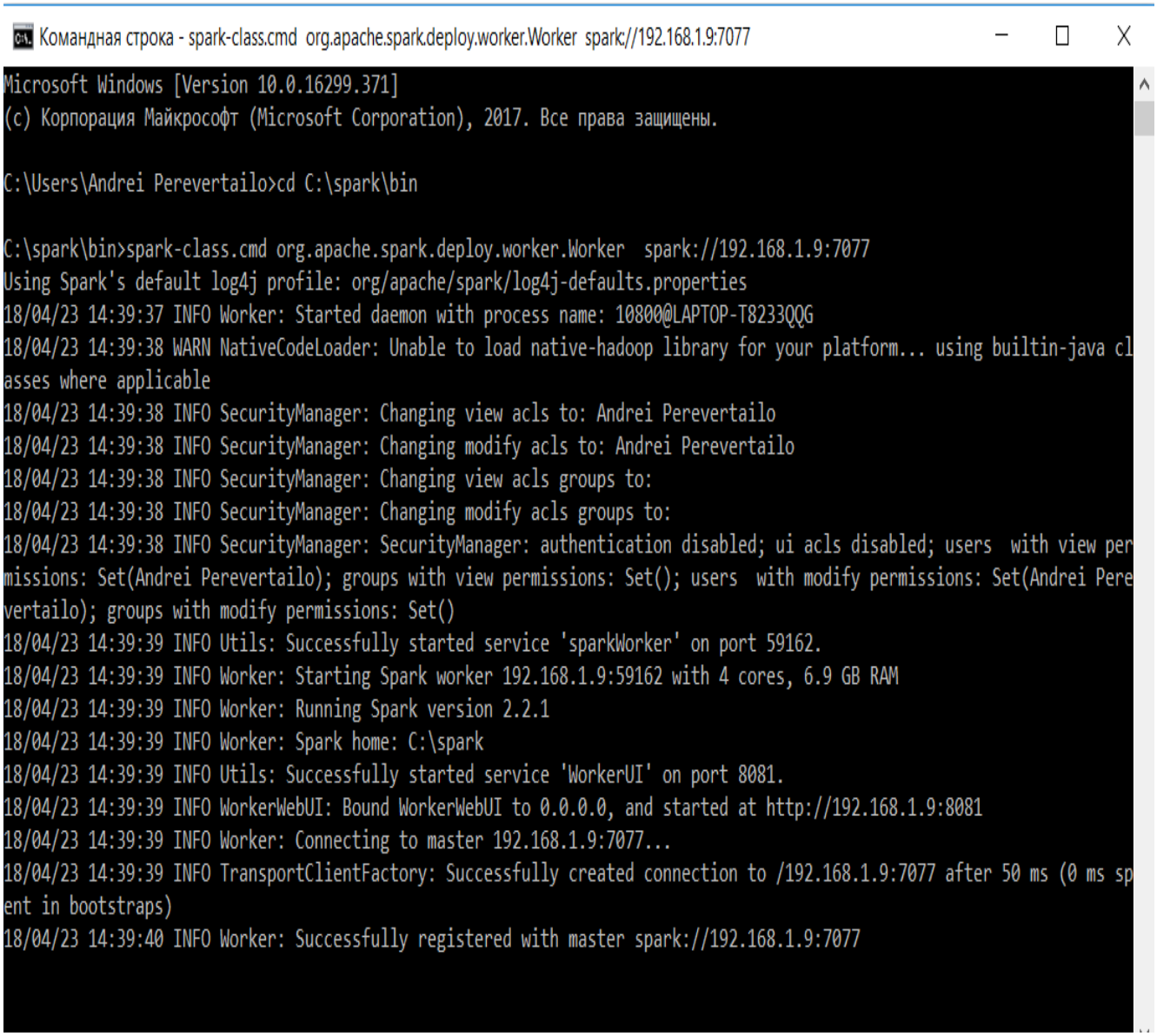

Рисунок 37 – Запуск підлеглого сервера засобами APACHE SPARK в операційній системі Windows

Використання платформи APACHE SPARK з метою утворення розподілених системи обробки даних дозволяє здійснювати автоматично розпаралелювання і завантаження даних на внутрішні диски вузлів кластера. Згадана вище архітектура приховує від програміста прикладні деталі внутрішнього стану APACHE SPARK і надає простий програмний інтерфейс, який забезпечує можливість

сконцентруватися над предметною логікою задачі, а не на її реалізації. APACHE SPARK надає можливість користувачам системи обробки та аналізу даних а також розробникам програмного забезпечення працювати з необмеженою кількістю типів даних із будь-яких джерел.

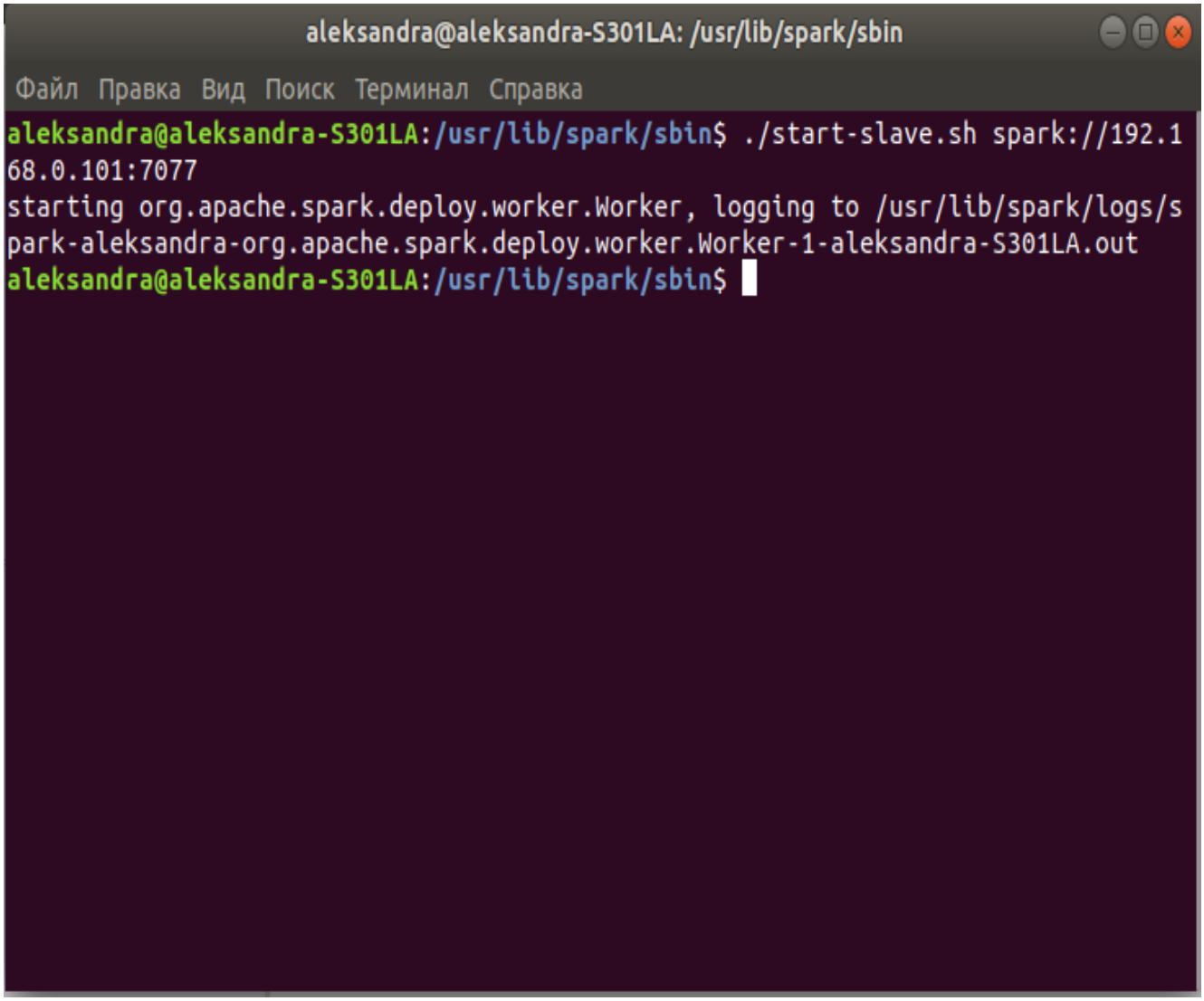

Рисунок 38 – Запуск підлеглого сервера засобами APACHE SPARK в операційній системі Ubuntu

SPARK Runtime Environment (SPARKEnv) - це середовище виконання з публічними службами SPARK, які взаємодіють один з одним для створення розподіленої обчислювальної платформи для програми SPARK. SPARK Runtime

Environment представлений об'єктом SPARKEnv, який містить всі необхідні служби виконання для запущеної програми SPARK з окремими середовищами для драйвера та виконавців.

На рисунку 39 зображено результати запуску підлеглих серверів, які разом із керуючим сервером утворюють обчислювальний кластер.

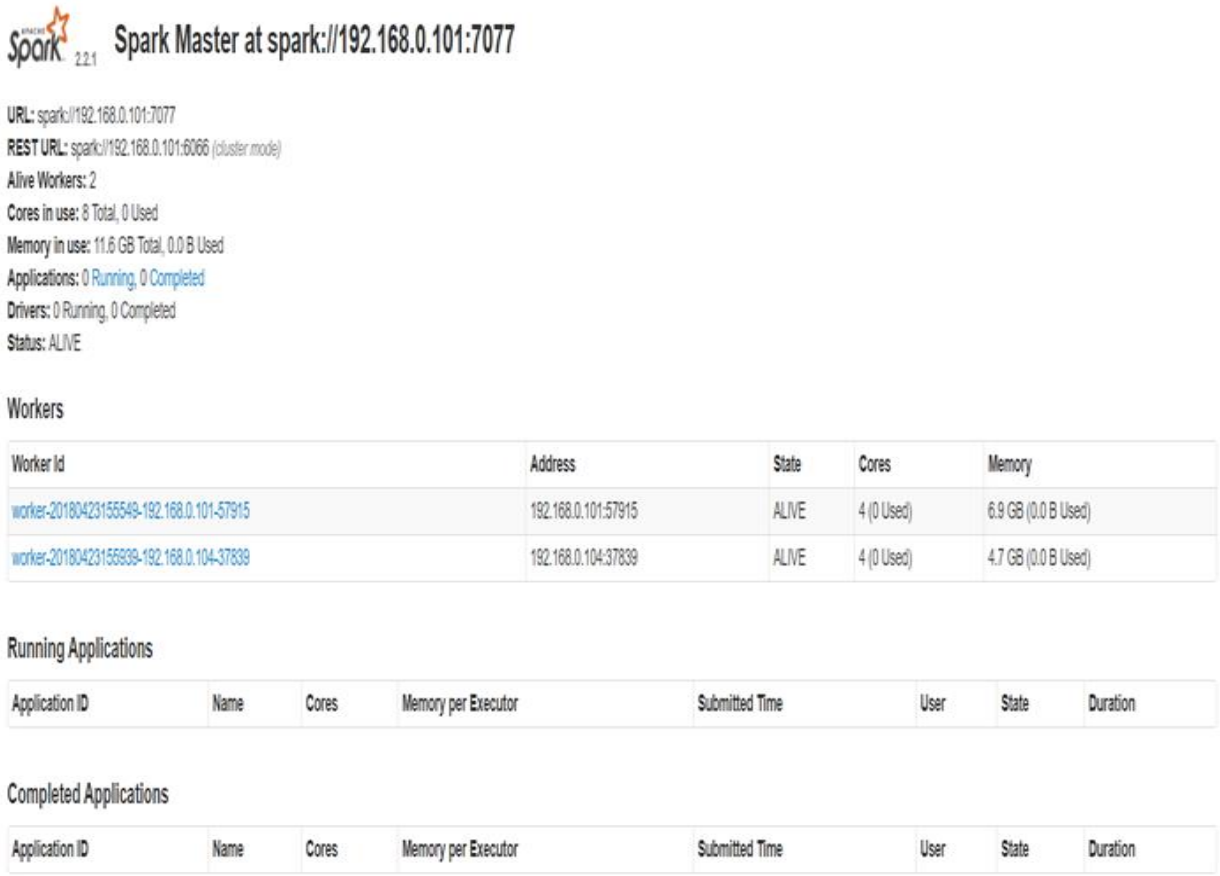

Рисунок 39 – Кластер, що з керуючого сервера та двох підлеглих серверів.

Виконавці (Executors) – розподілені агенти що відповідають за виконання програми на вузлах кластера. Виконавець створюється тоді, коли модуль CoarseGrainedExecutorBackend реєструє повідомлення про створення виконавця (рисунок 38) . Виконавець, як правило, працює протягом усього терміну дії програми SPARK, яка називається статичним розподілом виконавців (але ви можете також вибрати динамічне розміщення). Виконавці повідомляють про

виконання та часткові показники для активних задач серверу кінцевої точки RPC HeartbeatReceiver. Виконавці надають зберігання у пам'яті для RDD, кешовані в програмах SPARK (за допомогою диспетчера блоків). Виконавці можуть виконувати декілька задач протягом свого життя, як паралельно, так і повартоовно. Вони відвартоковують виконувані завдання (за їх ідентифікаторами задач у внутрішньому реєстрі runningTasks). Виконавці використовують пул потоків для запуску задач.

Виконавці описуються їхніми ідентифікаторами, іменами хостів, середовищем (як SPARKEnv) і classpath (більше для внутрішньої оптимізації, незалежно від того, чи вони працюють у локальному режимі або в кластері).

3.2 Запуск розробленої програми на кластері серверів

Після того як був налаштований обчислювальний кластер можна запустити розроблену програму. Запуск програми зображено на рисунку 40.

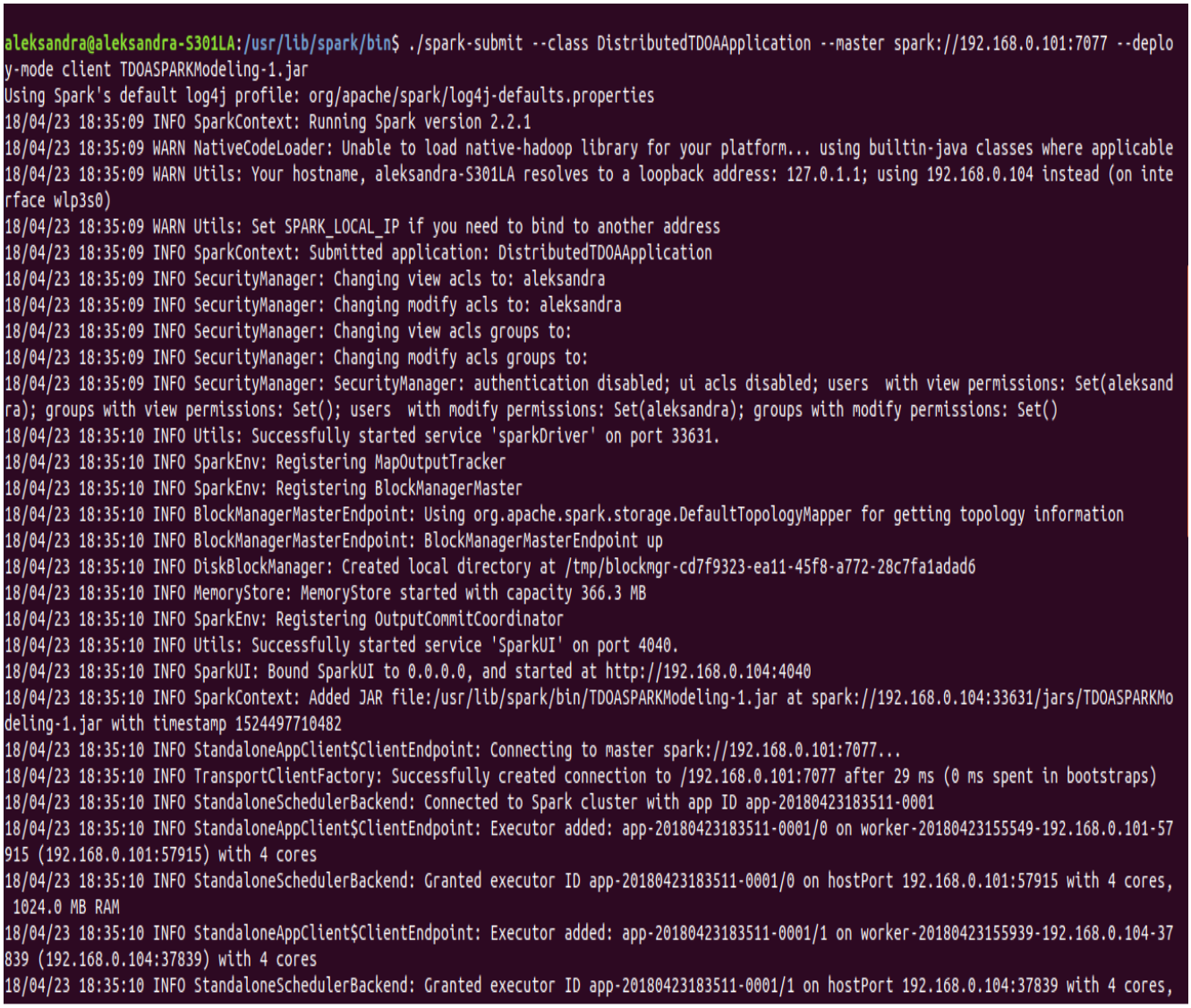

Рисунок 40 – Запуск підлеглого сервера засобами APACHE SPARK в операційній системі Ubuntu

Працівники (Workers) слугують підлеглими серверами на яких запускаються виконавці (Executors). Коли створюється SPARKContext, кожен працівник запускає виконавця(це окремий процес (JVM). Виконавці підключаються до програми. Тепер програма може надіслати їм команди, такі як flatMap, Map and reduceByKey. Коли програма завершується, виконавці закриваються.

Новий процес не починається для кожного кроку. Початок нового процесу для кожного працівника починається лише при побудові SPARKContext. Виконавець десеріалізує команду і виконує її в вузлі кластера.

Обчислювальні ресурси витрачені на виконання програми та час виконання програми можна побачити на рисунку 41.

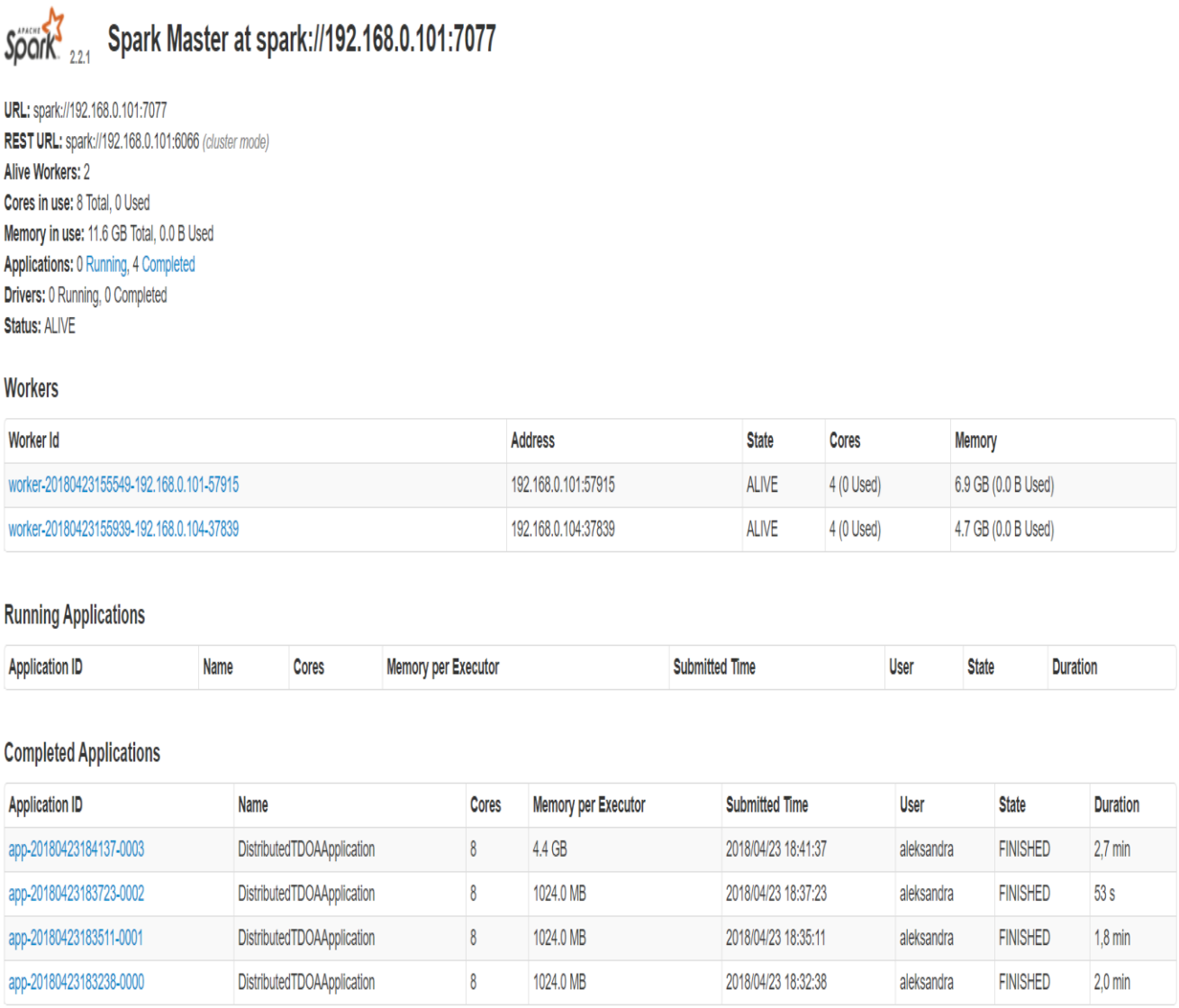

Рисунок 41. Статистика та час виконання програм з різною кількістю локалізованих джерел.

3.3 Висновки

У даному розділі сформовано вимоги щодо функціоналу програмного забезпечення, яке має бути встановлене на кожному вузлі кластера, продемонстровано компіляцію програми та продемонстровано запуск та налаштування керуючого та підлеглих серверів.

А також продемонстровано запуск розробленої програми на кластері серверів і проаналізовано результати запусків програми з різною кількістю джерел, місцеположення яких необхідно знайти.

4 АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

4.2 Порівняння кількості об’єктів, які можуть бути одночасно локалізовані без розподілу обчислень та в системі з розподіленим обчисленням координат

В ході роботи проаналізовано метод TDOA (Time Difference of Arrival) для локалізації об’єктів в розподілених сенсорних мережах. Побудовано модель, що імітує локалізацію джерел акустичних сигналів в сенсорній мережі та розподілену обробку даних сенсорів на вузлах кластера. Реалізовано алгоритм, що використовує метод TDOA (Time Difference of Arrival) та забезпечує розподілені обчислення на вузлах кластера засобами фреймворку APACHE SPARK.

Для довартоження можливості збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом проведено запуск моделі з різною кількістю джерел на одному комп’ютері та на кластері з двох комп’ютерів. Результати запуску продемонстровані у таблиці 2.

Таблиця 2 - Час обчислень координат на одному комп’ютері та на кластері з двох комп’ютерів.

Кількість джерел сигналів	Час обробки на одному комп’ютері, с	Час обробки на кластері з двох комп’ютерів, с
1000	74	100
3000	125	114
5000	180	120
6000	210	111

7000	320	168
8000	444	181
15000	-	304

З таблиці видно, що при більших навантаженнях розрахунок координат швидше виконується на кластері ніж на одному комп'ютері.

Це дає підстави стверджувати, що при збільшенні джерел сигналу до певної кількості або збільшенні кількості сенсорів доцільно організовувати кластер і динамічно масштабувати обчислювальні ресурси з метою збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом в розподіленій мережі сенсорів.

ВИСНОВКИ

В ході роботи проаналізовано архітектуру бездротових сенсорних мереж. Безпровідна сенсорна мережа скадається з мотів, будову мотів також проаналізовано, що з’єднані між собою та серверами, де відбуваються обчислення даних сенсорів. Були описані стандарт IEEE 802.15.4 та протокол ZigBee, засобами яких моти з’єднані між собою та серверами у мережу.

Також розглянуто метод TDOA (Time Difference of Arrival) для локалізації об’єктів в розподілених сенсорних мережах та алгоритм розрахунку координат джерела акустичного сигнала у мережі сенсорів. Наведені графіки визначення різниці часу приходу сигналів та ідентифікації джерела за сигналом.

Проаналізовано доцільність організації кластера серверів для обчислень даних серверів та вимоги до систем розподілених обчислень.

А також проаналізовано архітектуру сучасних платформи для обчислень великих колекцій даних. Вибрано платформу APACHE SPARK для подальшої розробки. Платформа APACHE SPARK відповідає вимогам зазначеним у пункті 1.3. Такий вибір обумовлений тим, що для задачі обчислення координат необхідно виконувати обробку даних в режимі реального часу, а платформа APACHE SPARK використовує оперативну пам’ять, а не дисковий простір, при обробці даних, що збільшує продуктивність розподілених обчислювальних систем побудованих на цій платформі.

Також вагомою перевагою э те, що SPARK складається з програми драйвера, яка запускає основну функцію користувача та виконує різні паралельні операції в кластері. Основна абстракція SPARK забезпечує еластичний розподілений набір даних (RDD), який являє собою набір елементів, розділених між вузлами кластера, які можуть працювати паралельно.

Отже, для обчислень даних сенсорів буде використовуватись кластер серверів під управлінням фреймворку APACHE SPARK.

Задача магістерської роботи полягає в оцінці можливості збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом в розподіленій мережі сенсорів.

Сформовано вимоги щодо функціоналу програмного забезпечення, що імітує локалізацію джерел акустичних сигналів в сенсорній мережі та складається з мережі сенсорів та серверів, що утворють кластер, а також використовує метод TDOA (Time Difference of Arrival) для локалізації сигналів та розподілює обчислення між серверами кластера реалізуючи модель APACHE SPARK.

Сформовано декілька етапів розподілу й обробки даних та проаналізовано трансформації вхідного потоку даних на вузлах кластера на кожному із етапів розподілу й обробки.

Створено UML діаграму класів та проведено опис призначення класів.

Сформовано вимоги щодо функціоналу програмного забезпечення, яке має бути встановлене на кожному вузлі кластера, продемонстровано компіляцію програми та продемонстровано запуск та налаштування керуючого та підлеглих серверів.

А також продемонстровано запуск розробленої програми на кластері серверів і проаналізовано результати запусків програми з різною кількістю джерел, місцеположення яких необхідно знайти.

Для довартоження можливості збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом проведено запуск моделі з різною кількістю джерел на одному комп’ютері та на кластері з двох комп’ютерів.

При більших навантаженнях розрахунок координат швидше виконується на кластері ніж на одному комп'ютері.

Це дає підстави стверджувати, що при збільшенні джерел сигналу до певної кількості або збільшенні кількості сенсорів доцільно організовувати кластер і динамічно масштабувати обчислювальні ресурси з метою збільшення кількості об’єктів, які можуть бути одночасно локалізовані за акустичним сигналом в розподіленій мережі сенсорів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Solution and performance analysis of geolocation by TDOA. [Електронний ресурс]. Режим доступу: https://www.hindawi.com/journals/isrn/2012/710979/.

2. TDOA Acoustic Localization. *Steven Li.* July 5, 2011. [Електронний ресурс]. Режим доступу: https://s3-us-west-1.amazonaws.com/stevenjl-bucket/TDOA_Localization.pdf.

3. Matrix Computations and Optimization in APACHE SPARK Режим доступу:

https://dl.acm.org/citation.cfm?id=2939675&picked=formats

4. TDOA denoising for acoustic source Localization. *M. Compagnoni, A. Canclini*. July 18, 2015 Режим доступу:

https://pdfs.semanticscholar.org/3513/2d7b099cff19b8315475f042373b079d5214.pdf

5. Topology Mapping algorithm for 2D and 3D Wireless Sensor Networks based on maximum likelihood estimation.

Ashanie Gunathillakeab Andrey V.Savkina Anura P.Jayasumanac May 5, 2017

Режим доступу: https://www.sciencedirect.com/science/article/pii/S1389128617303948

6. М. Беспроводные сенсорные сети [Електронний ресурс]. – Режим доступу: http://compress.ru/....

7. Judith B. Cardell Introduction to Wireless Sensor Networks [Електронний ресурс]. – Режим доступу: http://www.science.smith.edu/~jcardell/...

8. Akyildiz I.F., Su W., Sankarasubramaniam У., Cayirci E. Wireless sensor networks: a survey [Електронний ресурс]. – Режим доступу: http://citeseer.ist.psu.edu/...

9. Lewis F.L. Wireless Sensor Networks [Електронний ресурс]. – Режим доступу: http://210.32.200.159/....

10. Mark A. Perillo, Wendi B. Heinzelman Wireless Sensor Network Protocols [Електронний ресурс]. – Режим доступу: http://www.ece.rochester.edu/....

11. Незнамов Ю. Перспективы использования беспроводных ZigBee–интерфейсов в электроприводе [Електронний ресурс]. – Режим доступу: http://www.russianelectronics.ru/

12. Bilel Nefzi, Уе–Qiong Song PERFORMANCE ANALYSIS AND IMPROVEMENT OF ZIGBEE ROUTING PROTOCOL [Електронний ресурс]. – Режим доступу: http://hal.inria.fr

ДОДАТОК 1. Копії графічних матеріалів

ДОДАТОК 2. Лістинг програм

```
public class TDOAEngine {

	private String path1;

	private String path2;

	private String path3;

	private String path4;

	private String refer = "e:/data_refer.wav";


	private int[] beacon1;

	private int[] beacon2;

	private int[] beacon3;

	private int[] beacon4;


	private float[] result = new float[2];


	public TDOA(String[] path,int[][] beacon){

		this.path1 = path[0];

		this.path2 = path[1];

		this.path3 = path[2];

		this.path4 = path[3];


		this.beacon1 = beacon[0];

		this.beacon2 = beacon[1];

		this.beacon3 = beacon[2];

		this.beacon4 = beacon[3];


		localization();

	}
```

```
public float[] getResult(){
        return result;
}

private int timeDelayEstimation(String path){
        WaveFileReader wr1 = new WaveFileReader(path);
        WaveFileReader wr2 = new WaveFileReader(refer);
        float[] DataSig = wr1.getData()[0];
        float[] DataRefer = wr2.getData()[0];
        int lensig = wr1.getDataLen();
        int lenrefer = wr2.getDataLen();

        FFTprepare fpp = new FFTprepare(DataSig);
        float[] FFTsig = fpp.getsig();

        FFTcc fcc = new FFTcc(FFTsig,DataRefer);
        fcc.FindFccMax();
        float[] rcc = fcc.getrcc();
        float max = fcc.getmaxvalue();
        int maxplace = fcc.getmaxplace();

        int lenrcc = rcc.length;
        final float[] rccn;
        if(lensig<2400)
        {
                lensig = 4096;
        }
        if(lensig<=lenrcc)
```

```
		{
			rccn = new float[lensig];
			for(int i=lensig-1; i>=0; i--)
			{
				rccn[i] = rcc[lenrcc-1];
				lenrcc = lenrcc-1;
			}
		}
		else{
			rccn = rcc;
		}
		FindPeak fp = new FindPeak(rccn);
		int delay = fp.getPeakPlace();
		return delay;
	}

	private void localization(){
		int delay1 = timeDelayEstimation(path1);
		int delay2 = timeDelayEstimation(path2);
		int delay3 = timeDelayEstimation(path3);
		int delay4 = timeDelayEstimation(path4);

		float tdoa1 = ((float)(delay2-delay1)/44100)*340;
		float tdoa2 = ((float)(delay3-delay1)/44100)*340;
		float tdoa3 = ((float)(delay4-delay1)/44100)*340;

		Matrix a = new Matrix(new double[][]{
				{2*beacon1[0]-2*beacon2[0],2*beacon1[1]-2*beacon2[1],-
2*tdoa1},
```

```
                            {2*beacon1[0]-2*beacon3[0],2*beacon1[1]-2*beacon3[1],-
2*tdoa2},

                            {2*beacon1[0]-2*beacon4[0],2*beacon1[1]-2*beacon4[1],-2*tdoa3}

                });

                Matrix b = new Matrix(new double[][]{

                            {Math.pow(tdoa1,2)+Math.pow(beacon1[0],2)-
Math.pow(beacon2[0],2)+Math.pow(beacon1[1],2)-Math.pow(beacon2[1],2)},

                            {Math.pow(tdoa2,2)+Math.pow(beacon1[0],2)-
Math.pow(beacon3[0],2)+Math.pow(beacon1[1],2)-Math.pow(beacon3[1],2)},

                            {Math.pow(tdoa3,2)+Math.pow(beacon1[0],2)-
Math.pow(beacon4[0],2)+Math.pow(beacon1[1],2)-Math.pow(beacon4[1],2)}

                });

                Matrix aTranspose = a.copy().transpose();

                Matrix tmp = aTranspose.times(a);

                tmp = tmp.inverse();

                tmp = tmp.times(aTranspose);

                tmp = tmp.times(b);


                result[0] = (float)tmp.get(0,0);

                result[1] = (float)tmp.get(1,0);

        }

}
```